SUPPORT-THEORETIC SUBGRAPH PRECONDITIONERS FOR LARGE-SCALE SLAM AND STRUCTURE FROM MOTION

A Dissertation
Presented to
The Academic Faculty

by

Yong-Dian Jian

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Interactive Computing
College of Computing

Georgia Institute of Technology
August 2014

SUPPORT-THEORETIC SUBGRAPH PRECONDITIONERS FOR LARGE-SCALE SLAM AND STRUCTURE FROM MOTION

Approved by:

Professor Frank Dellaert, Advisor
School of Interactive Computing
College of Computing
Georgia Institute of Technology

Professor Edmond Chow
School of Computer Science and
Engineering, College of Computing
Georgia Institute of Technology

Professor James M. Rehg
School of Interactive Computing
College of Computing
Georgia Institute of Technology

Professor Prasad Tetali
School of Computer Science and
School of Mathematics
Georgia Institute of Technology

Professor Noah Snavely
Department of Computer Science
Cornell University

Date Approved: May 29, 2014

To the people have faith in me,

ACKNOWLEDGEMENTS

First I would like to express my sincere gratitude to my advisor, Frank Dellaert. His knowledge, sense of humor, hard working, and unique ways of thinking and writing always inspire me and establish an ideal model of a scholar in my mind.

I am also grateful to many researchers that I met and worked with. I would like to thank Edmond Chow for providing helpful feedbacks to my thesis work, James Rehg for making the big picture of this dissertation as well as the job advice, Prasad Tetali and Doru Balcan for the helpful discussions on formulating and solving the core problem in my thesis, Noah Snavely for the insightful comments from the large-scale SfM perspective, Blair MacIntyre for supporting my initial work on mobile tracking and augmented reality, Harris Bergman, Igor Kozintsev and Yuichi Taguchi for hosting my internships and providing collaboration opportunities. I also want to thank NSF Award 11115678 to support my research in the past three years.

Many thanks to my colleagues that I can always count on for sharing my ideas and thoughts, especially to Pablo Alcantarilla, Chris Beall, Luca Carlone, Changhyun Choi, Siddarth Choudhary, Alex Cunningham, Jing Dong, Can Erdogan, Alireza Fathi, Viorela Ila, Vadim Indelman, Abhijit Kundu, Andrew Melim, Kai Ni, Carlos Nieto, Sang Min Oh, Kyle Ok, Manohar Paluri, Ioannis Panageas, Richard Roberts, Grant Schindler, Natesh Srinivansan, Duy-Nguyen Ta, David Tsai, and Steven Williams.

Finally I want to thank my family for their support, especially to my wife, Yu-Ying, whose love and companionship gives me the strength and wisdom to move forward through the challenges. The last but not the least, I would like to thank our hilarious daughter, Jasmine, whose arrival to this world not only gives a new meaning to my life, but also accelerates the completion of this dissertation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Simultaneous localization and mapping (SLAM) and Structure from Motion (SfM) are important problems in robotics and computer vision. One of the challenges is to solve a large-scale optimization problem associated with all of the robot poses, camera parameters, landmarks and measurements. Yet neither of the two reigning paradigms, direct and iterative methods, scales well to very large and complex problems. Recently, the subgraph-preconditioned conjugate gradient method has been proposed to combine the advantages of direct and iterative methods. However, how to find a good subgraph is still an open problem.

The goal of this dissertation is to address the following two questions: (1) What are good subgraph preconditioners for SLAM and SfM? (2) How to find them? To this end, I introduce support theory and support graph theory to evaluate and design subgraph preconditioners for SLAM and SfM. More specifically, I make the following contributions:

First, I develop graphical and probabilistic interpretations of support theory and used them to visualize the quality of subgraph preconditioners.

Second, I derive a novel support-theoretic metric for the quality of spanning tree preconditioners and design an MCMC-based algorithm to find high-quality subgraph preconditioners. I further improve the efficiency of finding good subgraph preconditioners by using heuristics and domain knowledge available in the problems. Our results show that the support-theoretic subgraph preconditioners significantly improve the efficiency of solving large SLAM problems.

Third, I propose a novel Hessian factor graph representation, and use it to develop

a new class of preconditioners, generalized subgraph preconditioners, that combine the advantages of subgraph preconditioners and Hessian-based preconditioners. I apply them to solve large SfM problems and obtain promising results.

Fourth, I develop the incremental subgraph-preconditioned conjugate gradient method for large-scale online SLAM problems. The main idea is to combine the advantages of two state-of-the-art methods, incremental smoothing and mapping, and the subgraph-preconditioned conjugate gradient method. I also show that the new method is efficient, optimal and consistent.

To sum up, preconditioning can significantly improve the efficiency of solving large-scale SLAM and SfM problems. While existing preconditioning techniques do not utilize the problem structure and have no performance guarantee, I take the first step toward a more general setting and have promising results.

CHAPTER I

INTRODUCTION

Large-scale mapping is an important problem in robotic and computer vision. In robotics, the problem is referred to as *Simultaneous Localization and Mapping* (SLAM) [33, 8], which aims to build a map of an unknown environment while at the same time keeping track of the robot's current location. An efficient and robust solution to the SLAM problem has been considered as a "holy grail" in the robotics community because it leads to many applications for autonomous robots.

In computer vision, the problem is referred to as *Structure from Motion* (SfM) [45], which aims to reconstruct the scene structure, camera poses, and camera parameters by using the correspondences from an *unstructured* collection of images.

The common computational bottleneck of SLAM and SfM is to solve a large nonlinear optimization problem. The state-of-the-art solvers stem from the preconditioned conjugate gradient (PCG) method. Crucial to the performance of PCG is the choice of preconditioner, but existing preconditioning techniques such as the block-Jacobi or incomplete factorization preconditioners are not satisfactory because they do not utilize the problem structure and provide limited performance guarantee.

1.1 Thesis Statement

My thesis statement is as follows:

Support-theoretic subgraph preconditioners are novel and effective preconditioners for solving large-scale SLAM and Structure from Motion problems.

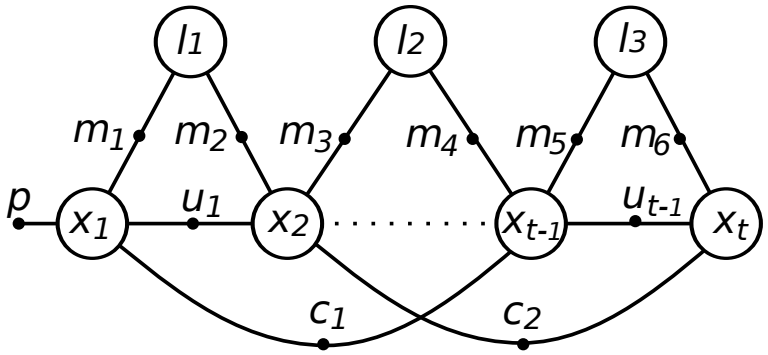


Figure 1: The factor graph formulation of a simple SLAM problem, where the unknowns are shown as circles, and the factors (measurements) are shown as solid dots. The factors can denote priors p, odometry measurements u, and landmark measurements m, and loop-closure constraints c. Special cases include the pose-graph formulation (without l and m), and landmark-based SLAM (without c).

1.2 Simultaneous Mapping and Localization (SLAM)

Simultaneous localization and mapping (SLAM) refers to the problem of localizing a robot in an unknown environment while simultaneously building a consistent map of the environment [93, 73, 101, 33, 8]. Being able to conduct SLAM in large and complex environments is important for autonomous mobile robots. Figure 1 shows the graphical model [61] of a simple SLAM problem.

In the last two decades, significant progress has been made to solve the SLAM problem. Existing approaches can be grouped into two categories. The first category consist of the *online* approaches, which assume the measurements come in an online fashion. The second category consists of the *batch* approaches, which assume the measurements are available all at once.

The earlier works on the online approaches are based on the extended Kalman filter (EKF) [92, 73], which recursively estimates a Gaussian distribution of the current robot state and the landmarks. Yet the computational complexity of these approaches may grow quadratically which make them unsuitable for large-scale problems. Many works extended the filtering-based approach to cope with the scalability problem. Yet the filtering-based approaches have been shown to be inconsistent when applied

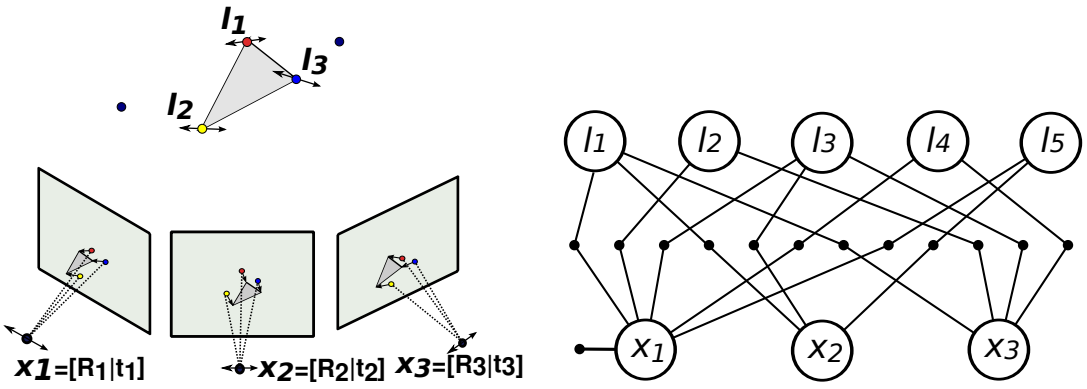


Figure 2: A simple SfM problem with three cameras and five points. On the left is the physical configuration. On the right is the corresponding factor graph representation.

to solve nonlinear SLAM problems [57]. Many works have also been proposed to improve EKF-based methods, e.g., information filters [36, 100], particle filters [79, 80], Graphical SLAM [38], and iSAM [59, 58].

Recently, there has been a considerable interest in the batch version of the SLAM problem [77, 100, 99, 29, 85, 43, 42, 81]. These approaches take into account all of the available measurements and generate the optimal estimates at once. Because all of the information is available beforehand, the batch approaches can produce better strategies than the online approaches [98]. In this dissertation, I will focus on developing more efficient solutions for both batch and online SLAM problems.

1.3 Structure from Motion (SfM)

Structure from Motion (SfM) refers to the problem of recovering the scene structure and the camera motions by using the correspondences among a collection of unstructured images [45, 103, 89, 37, 97, 94, 25]. Figure 2 shows a simple SfM problem.

SfM has become very useful due to the ubiquitousness of digital cameras. Many systems have been built to exploit the internet photo collections. The most visible efforts are the "Photo Tourism" and "Build Rome in a Day" projects [95, 4].

SfM is typically solved in two stages. In the first stage, certain types of features

such as points, lines and planes are extracted and matched in the images. In the second stage, a large nonlinear optimization problem referred to as *bundle adjustment* (BA) has to be solved to obtain the optimal estimates [105]. In this dissertation, I focus on developing more efficient solutions for this optimization problem.

1.4 Challenges in SLAM and SfM

Although appearing in different domains, SLAM and SfM have several challenges in common. One of them is to solve a graph-based nonlinear least-squares problem where the vertices denote the robot/camera poses and scene structure, and the edges represent the squared error terms associates with the measurements. The main difference is in their graph structures. In SLAM, the graph structure depends on the robot's trajectory and the possessed sensors. In SfM, the graph is typically an unbalanced bipartite graph of cameras and 3D points.

Despite their difference in the graph structures, these two problems share one common computational bottleneck, which is to solve a large, sparse and typically ill-conditioned linear system. Yet neither of the two reigning paradigms, direct and iterative methods, scales well to very large and complex problems. The state-of-the-art solvers stem from the preconditioned conjugate gradient (PCG) method [69, 30, 5, 70]. Crucial to the performance of PCG is the choice of preconditioner, but generic preconditioning techniques such as the block-Jacobi preconditioner or incomplete factorization preconditioner [87] are not satisfactory because they do not utilize the problem structure and provide limited performance guarantee.

Recently, the subgraph-preconditioned conjugate gradient method has been proposed to combine the advantages of direct and iterative methods, and it has demonstrated promising performance to solve large-scale SLAM problems [30]. However, how to find a good subgraph is still an open problem, and this motivates the development of this dissertation.

1.5 Overview of the Dissertation

The dissertation is organized as follows. First, I will review the problem formulation and related work in Chapter 2. The goal of this dissertation is to find good subgraph preconditioners to improve the efficiency of solving large SLAM and SfM problems. I have made the following contributions:

- I develop graphical and probabilistic interpretations of support theory [15] and applied them to intuitively visualize the preconditioning process as well as the quality of subgraph preconditioners (Chapter 3). I also compare the performance of subgraph preconditioners with two standard preconditioning techniques on simulated SLAM problems (Chapter 4).

- I derive a support-theoretic metric for the quality of spanning tree preconditioners and designed an MCMC-based algorithm to find high-quality subgraph preconditioners [52]. I then apply the heuristics and domain knowledge to improve the efficiency of finding good subgraphs. The results show that support-theoretic subgraph preconditioners significantly improve the efficiency of solving large SLAM problems (Chapter 5).

- I propose a new class of preconditioners, generalized subgraph preconditioners, which combine the advantages of subgraph preconditioners and Hessian-based preconditioners, such as the Jacobi preconditioner. I apply them to solve large SfM problems and obtained promising results [53, 54] (Chapter 6).

- I extend our technique to the online scenario and develop a new method, incremental subgraph-preconditioned conjugate gradient method, that combines two state-of-the-art methods and delivers promising performance for online large-scale SLAM problems. I also show that the new method is efficient, optimal and consistent [55] (Chapter 7).

CHAPTER II

RELATED WORK

2.1 Problem Formulation

Here I review a unified problem formulation for SLAM and SfM to facilitate the exposition. Let us define $\boldsymbol{\theta} = \{\theta_i\}_{i=1}^n$ as the state variables (e.g., robot poses, camera poses, 3D points), and $\mathbf{Z} = \{z_j\}_{j=1}^m$ as the measurements. The goal is to obtain the maximum a posteriori (MAP) estimation

$$\boldsymbol{\theta}_{\text{MAP}}(\mathbf{Z}) = \underset{\boldsymbol{\theta}}{\text{argmax}}\, P(\boldsymbol{\theta})P(\mathbf{Z} \mid \boldsymbol{\theta}). \tag{1}$$

Assuming the variables are independent, and the measurements are conditionally independent, we can factorize the right-hand side of (1) into

$$P(\boldsymbol{\theta})P(\mathbf{Z}|\boldsymbol{\theta}) \propto \prod_{i=1}^{n} P(\theta_i) \prod_{j=1}^{m} P(z_j \mid \theta_j) \tag{2}$$

where θ_j denotes the set of variables associated with the jth measurement.

The SLAM and SfM problems can also be formulated with the factor graph [68] where each vertex denotes a state variable, and each factor (edge) denotes the squared error term associated with a probability density function in (2). More specifically, I assume the prior and measurement models are Gaussian, defined by

$$P(\theta_i) \propto \exp(-\|g_i(\theta_i)\|^2_{\Gamma_i}) \tag{3}$$

$$P(z_j \mid \theta_j) \propto \exp(-\|h_j(\theta_j)\|^2_{\Psi_j}). \tag{4}$$

where $g_i(\cdot)$ denotes the prior model over the ith variable and $h_j(\cdot)$ denotes the model of the jth measurement. In both models, I assume zero-mean and normally distributed noise with covariance matrices Γ_i and Ψ_j respectively. Here $\|e\|_\Sigma = \sqrt{e^T \Sigma^{-1} e}$ denotes the Mahalanobis distance. By substituting the probability densities in (2) with the

functions in (3) and (4), and taking negative logarithm, we obtain the following factor graph representation of (2)

$$\boldsymbol{\theta}_{\mathrm{MAP}}(\mathbf{Z}) = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{i=1}^{n} \|g_i(\theta_i)\|^2_{\Gamma_i} + \sum_{j=1}^{m} \|h_j(\theta_j)\|^2_{\Psi_j}. \tag{5}$$

More generally, we can combine the two terms in (5) as

$$\boldsymbol{\theta}_{\mathrm{MAP}}(\mathbf{Z}) = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{k=1}^{m+n} \|e_k(\theta_k)\|^2_{\Sigma_k} \tag{6}$$

where $e_k(\cdot)$ denotes a function on a set of variables θ_k with a covariance matrix Σ_k.

2.2 Nonlinear Optimization Approach

Here I show how to solve (6) via the nonlinear optimization approach. In general, the function in (6) is not convex and has no closed-form expression to compute the optimum, but assuming we have some initial estimates of the variables, we can find a local minimum by using any nonlinear least-squares optimization technique such as the Gauss-Newton or the Levenberg-Marquardt algorithm [84].

The key is to apply the first-order Taylor expansion to linearize the function as

$$e_k(\theta_k) \approx e_k(\theta_k^0) + \mathbf{J}_k \Delta\theta_k \tag{7}$$

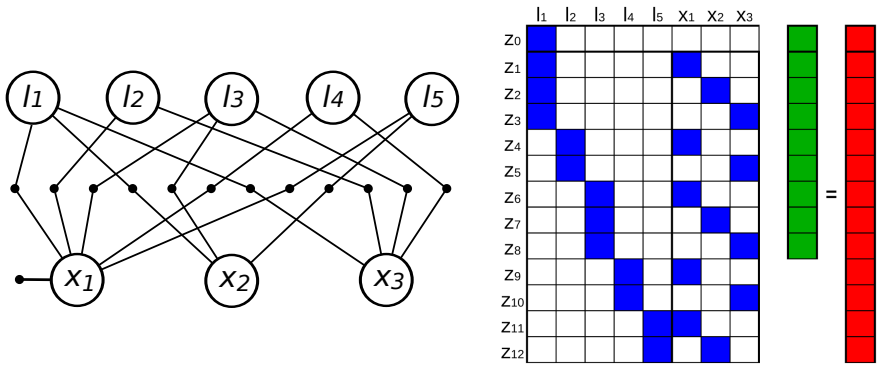
where $\mathbf{J}_k$ is the Jacobian matrix of $e_k(\cdot)$ with respect to θ_k at a linearization point θ_k^0:

$$\mathbf{J}_k = \left.\frac{\partial e_k(\theta_k)}{\partial \theta_k}\right|_{\theta_k^0}. \tag{8}$$

If we set (7) to zero, then we obtain $\mathbf{J}_k \Delta\theta_k = -e_k(\theta_k^0)$ which is linear in $\Delta\theta_k$. Repeating this procedure for all of the $e_k(\cdot)$ functions, we can derive a linear system

$$\mathbf{A}\Delta\boldsymbol{\theta} = \mathbf{b} \tag{9}$$

where $\mathbf{A}$ is a rectangular matrix whose kth (block) row contains the Jacobian matrix $\mathbf{J}_k$ in (8), and $\mathbf{b}$ is a vector whose kth (block) row equals $-e_k(\theta_k^0)$.



(a) (b)

Figure 3: An SfM example with its (a) factor graph and (b) matrix representations.

Equation (9) can be considered as a *linearized* version of the unified problem whose graph structure is represented by the sparsity pattern of the matrices, e.g., see Figure 3. Hereafter I will refer to $\mathbf{A}$ as the *Jacobian matrix*, and (9) as the *linear system* or the *Gaussian factor graph* of the unified problem. We can iteratively solve (9) to update the current estimates until convergence.

An alternative way to compute $\Delta\boldsymbol{\theta}$ is to solve the *normal equations*

$$(\mathbf{A}^T\mathbf{A})\Delta\boldsymbol{\theta} = \mathbf{A}^T\mathbf{b}, \tag{10}$$

where $(\mathbf{A}^T\mathbf{A})$ is a first-order approximation to the Hessian matrix of (6). Solving this equation is more efficient when the rows of $\mathbf{A}$ is much more than the columns of $\mathbf{A}$. Hereafter I will refer to $\mathbf{A}^T\mathbf{A}$ as the (approximate) *Hessian matrix*.

To sum up, solving the unified problem (6) is equivalent to solving a sequence of sparse linear systems in (9) or (10). In the following two sections, I will briefly review the two reigning paradigms of solving sparse linear systems.

2.3 Direct Methods

Direct methods work by transforming a matrix into a triangular matrix (row echelon form), followed by a forward/backward substitution step to compute the solution. For instance, we can use QR factorization to solve (9), or use Cholesky factorization

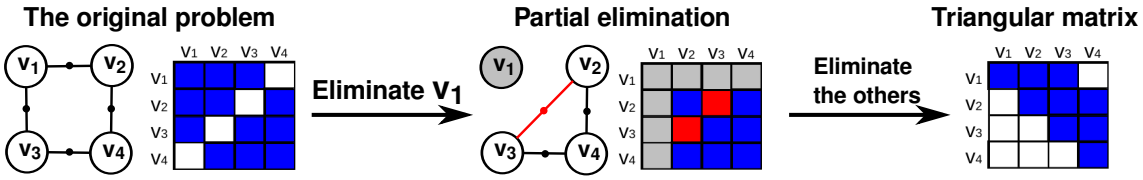


Figure 4: An example of how direct methods work.

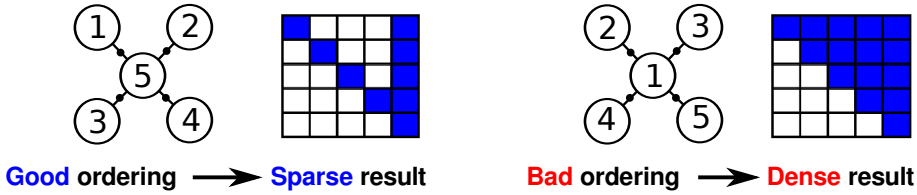


Figure 5: An example of how the elimination ordering affects the sparsity of the triangular matrix.

to solve (10) [104, 26]. More specifically, given a linear system

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{11}$$

where $\mathbf{A} \in \mathbb{R}^{m \times n}$ is a rectangular matrix and $\mathbf{b} \in \mathbb{R}^m$ is a vector, then $\mathbf{A}$ can be decomposed as

$$\mathbf{A} = \mathbf{Q}\mathbf{R} \tag{12}$$

where $\mathbf{Q} \in \mathbb{R}^{m \times n}$ is an orthonormal matrix and $\mathbf{R} \in \mathbb{R}^{n \times n}$ is an upper triangular matrix. Then we can compute the solution as

$$\mathbf{x} = \mathbf{R}^{-1}\mathbf{Q}^T\mathbf{b}. \tag{13}$$

If $\mathbf{A}$ is a symmetric and positive definite, then it can be decomposed as

$$\mathbf{A} = \mathbf{R}^T\mathbf{R}, \tag{14}$$

and we can obtain the solution via the following equation

$$\mathbf{x} = \mathbf{R}^{-1}\mathbf{R}^{-T}\mathbf{b}. \tag{15}$$

Direct methods can be best explained as a sequence of variable eliminations on factor graph when the matrices are sparse. Each time we eliminate a variable (vertex),

a new factor connecting to all of its neighbors will be added to the graph. After eliminating all of the variables, we obtain an upper triangular matrix as a result. Figure 4 illustrates the variable elimination process.

The variable elimination ordering is important to the efficiency of direct methods. Using a good ordering would induce less fill-in and lead to a sparse triangular matrix, which requires less storage and also makes the forward and backward substitutions more efficient. Figure 5 shows how the ordering affects the sparsity of the factorized matrix. Yet finding the optimal ordering that leads to the least fill-in is an NP-hard problem [86]. Many heuristic algorithms to find good elimination ordering have been developed [7, 28].

In SLAM, the use of direct methods was first suggested in [77, 29]. A common approach to reduce the computational cost is to split a large problem into smaller subproblems. In these approaches [42, 81], local maps are limited to a bounded number of poses and landmarks. Therefore the local maps can be constructed in constant time, but additional computation is shifted to the map joining phase.

In SfM, the use of direct methods was first suggestion by Brown [17]. Due to the unbalanced bipartite graph structure, it is more efficient to eliminate all of the 3D points first and use direct methods to solve the *reduced camera system* [105, 76, 62]. The idea of splitting the problem into smaller subproblems is also explored [82]

While direct methods are efficient for *sparse* problems, they may require storage quadratically proportional to the problem size when there are many loop-closures or complex graph structures. This property makes direct methods impractical for large-scale problems. To resolve this problem, several recent works suggested using *iterative methods* which will be summarized in the next section.

2.4 Iterative Methods

Iterative methods work by generating a sequence of improving approximate solutions. They are better alternative to direct methods for large problems because they do not introduce fill-in and perform only simple operations, but they may suffer from slow convergence if the original problem is ill-conditioned.

Iterative methods have been successfully applied to solve *nonlinear* SLAM and SfM problems (6). Their application to SLAM was pioneered by [32] and [48], who used the Gauss-Seidel relaxation on the constraint graph. Frese et al. [39] presented a multi-level relaxation algorithm, inspired by the multigrid method [16]. The state-of-the-art SLAM approach stems from the work of Olson et al. [85], whose main contribution is to re-parametrize the global poses in terms of the increments along the robot's trajectory, and it leads to faster convergence for the stochastic gradient decent method. Grisetti et al. [43] adapted this idea to re-parametrize the poses with a spanning tree of the original graph.

Iterative methods have also been applied to solve *linearized* SLAM and SfM problems (9) [63, 30, 69, 105, 5, 51, 53, 70]. Among all of the iterative methods, the conjugate gradient (CG) method is the method of choice because of its efficiency and minimization property. Please refer to Appendix A for the mathematical background.

Yet the plain CG method could still be inefficient if the problem is *ill-conditioned*, meaning the ratio between the extreme singular values of $\mathbf{A}$ is large large. Therefore the original linear least-squares problem has to be *reparametrized* to become a well-conditioned problem. This is the main idea behind *preconditioning*.

2.5 Preconditioning

Preconditioning refers to a procedure of transforming a problem into a form that can be solved more efficiently by iterative methods. The importance of preconditioning cannot be over-emphasized. Here I quote Trefethen and Bau [104]:

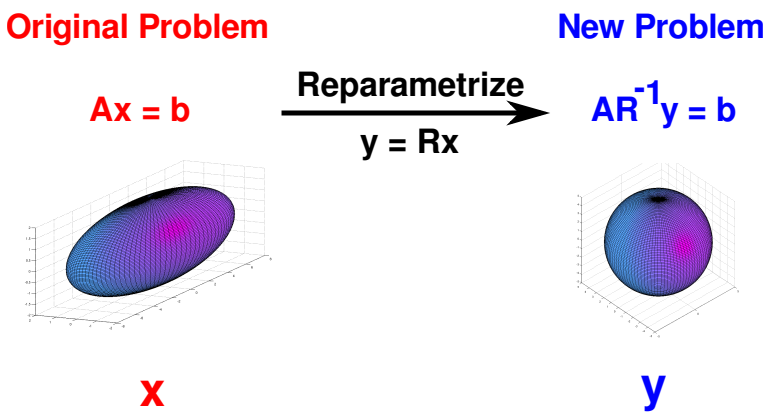


Figure 6: The preconditioning process.

> Nothing will be more central to computational science in the next century than the art of transforming a problem that appears intractable into another whose solution can be approximated rapidly. For [iterative methods], this is preconditioning.

There is a least-squares variant of the preconditioning conjugate gradient (PCGLS) method, which aims to solve a preconditioned linear least-squares problem

$$\mathbf{A}\mathbf{R}^{-1}\mathbf{y} = \mathbf{b} \tag{16}$$

where $\mathbf{R}$ is called the *preconditioner*. After using PCGLS to find the solution of the preconditioned system, we can recover the solution of the original problem via $\mathbf{x} = \mathbf{R}^{-1}\mathbf{y}$. This can also be interpreted as variable reparametrization. Figure 6 illustrates the preconditioning process. The detail of PCGLS can be found in Appendix A.

Conditions 1. *A good preconditioner has to satisfy the following two conditions:*

(1) $\mathbf{R}^{-1}$ *can be computed and applied efficiently, and*

(2) $(\mathbf{R}^{-T}\mathbf{A}^T)(\mathbf{A}\mathbf{R}^{-1})$ *has clustered eigenvalues or the generalized condition number* $\kappa(\mathbf{A}^T\mathbf{A}, \mathbf{R}^{-T}\mathbf{R}^{-1})$ *is small.*

Many preconditioning techniques have been developed in literature, but most of them are designed to satisfy the first condition, and provide poor or no guarantee on

the second condition. For example, the block Jacobi preconditioner works by using the block diagonal of $\mathbf{A}^T\mathbf{A}$ [87] so that the preconditioner can be computed and applied efficiently. The quality of the block Jacobi preconditioner is inversely proportional to the block size. To improve the quality of preconditioner, one has to increase the block size but this will eventually destroy the efficiency of this technique. Another popular preconditioning technique is based on the incomplete matrix factorization [87, 12]. These techniques emphasize on satisfying the first condition by using heuristics to control the number of nonzeros in the preconditioner during the matrix factorization phase. However, the second condition is not explicitly enforced.

For SLAM, the use of the preconditioned conjugate gradient method with the incomplete Cholesky preconditioner was suggested by Konolige [63]. Dellaert et al. [30] used subgraph preconditioners and reported promising results. Recently, Kümmerle et al. [69] integrated the block-Jacobi preconditioner into an optimization library.

For SfM, Agarwal et al. [5] examined the performance of several standard preconditioners and implementation strategies on large-scale datasets. Byröd and Åström [18, 19] proposed to use multi-scale and the block Jacobi preconditioners respectively. Jeong et al. [51] suggested using the band-diagonal of the reduced camera system as a preconditioner. Kushal and Agarwal [70] used the visibility patterns to build cluster diagonal and cluster tridiagonal preconditioners along with the modified incomplete Cholesky factorization. Yet all of the above preconditioning techniques focus on the first condition without guarantee on the second condition.

2.6 Subgraph Preconditioners

In this dissertation, I am interested in using a particular class of preconditioners, *subgraph preconditioners*, to improve the efficiency of solving the unified problem. The main idea is to use a *sparse* sub-problem (subgraph) to build a preconditioner for the original problem (graph) [71, 12, 30, 53] (see Figs. 7 and 8). I will focus on

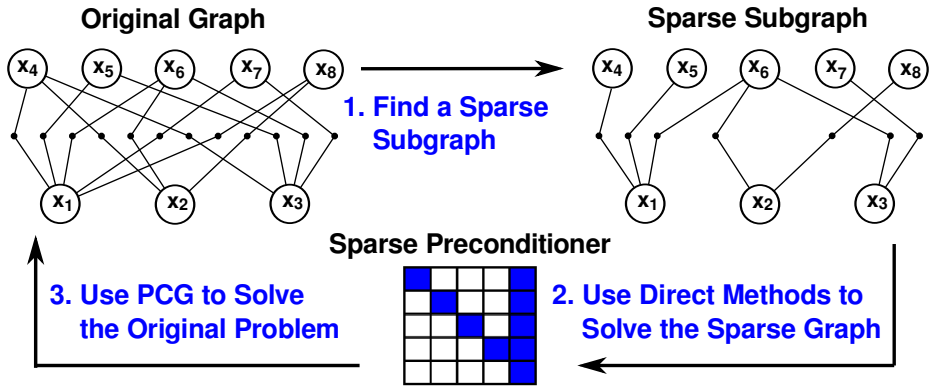


Figure 7: The idea of subgraph preconditioners.

presenting the formulation of subgraph preconditioners. The theoretical properties of subgraph preconditioners will be discusses in the next section.

Here I present the subgraph preconditioners in the context of the CGLS method, but it can also be used with the PCG or the PCGLS method. For any linear least-squares problem or Gaussian factor graph

$$f(\mathbf{x}) = \|\mathbf{A}_G\mathbf{x} - \mathbf{b}_G\|^2, \tag{17}$$

we first identify a sparse sub-problem (subgraph) $f'(\mathbf{x}) = \|\mathbf{A}_H\mathbf{x} - \mathbf{b}_H\|^2$ where $(\mathbf{A}_H, \mathbf{b}_H)$ corresponds to a subset of rows in $(\mathbf{A}_G, \mathbf{b}_G)$. The subscript G denotes the original graph while the subscript H denotes a subgraph of G. Hence we can rewrite (17) as

$$f(\mathbf{x}) = \|\mathbf{A}_H\mathbf{x} - \mathbf{b}_H\|^2 + \|\mathbf{A}_C x - \mathbf{b}_C\|^2 \tag{18}$$

where the subscript $C = G \backslash H$ denotes the complement of H. Finally we use any sparse direct solver to factorize $\mathbf{A}_H$ to build the preconditioner.

There are two ways to apply the subgraph preconditioners. The first is to plug the preconditioner into the PCGLS method (see Algorithm 4), but it may not be the most efficient implementation.

The second way is to explicitly use the preconditioner to re-parametrize the problem. Consider applying QR factorization to $\mathbf{A}_H = \mathbf{Q}_H\mathbf{R}_H$ to obtain the solution

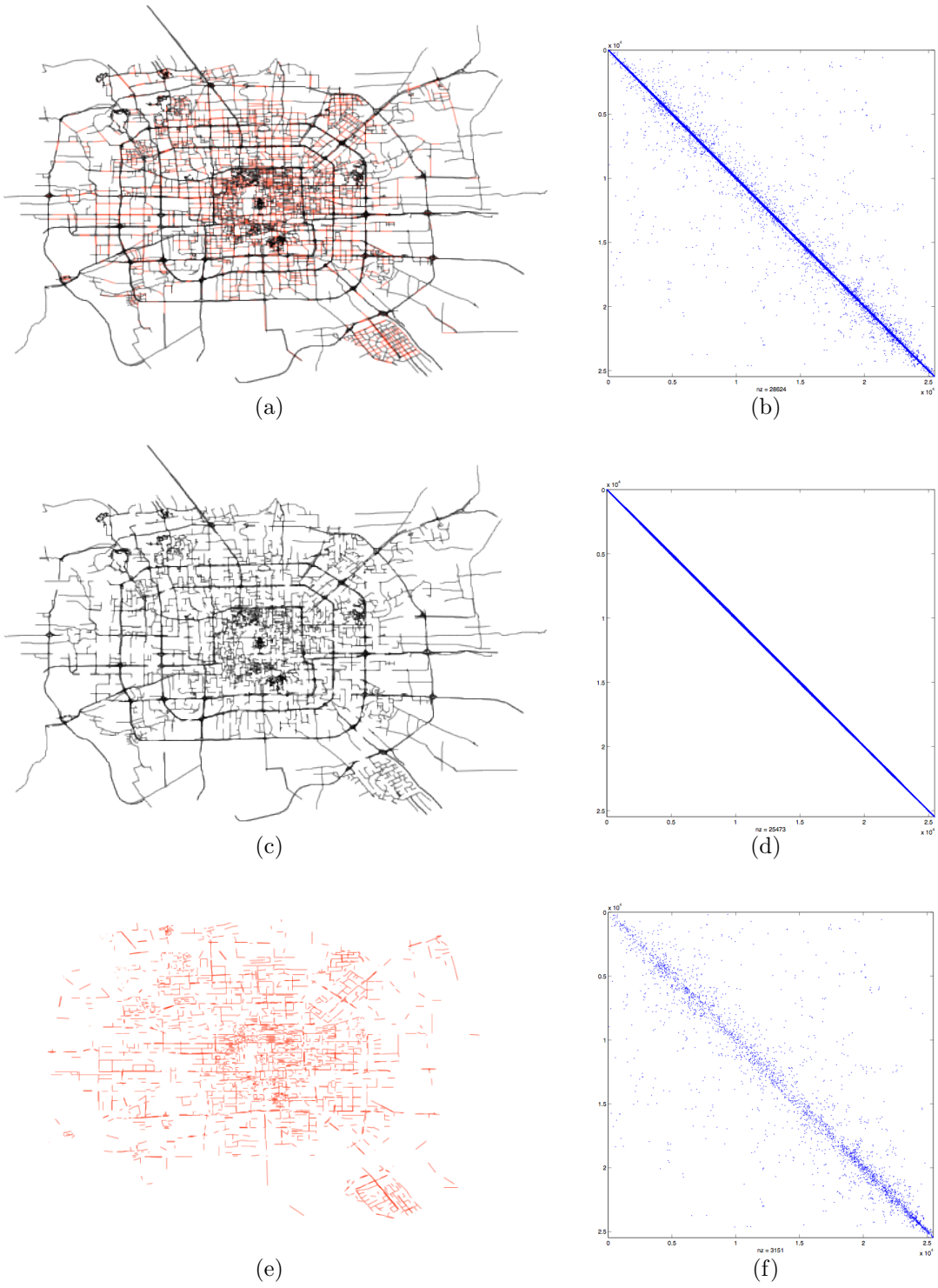


Figure 8: Illustration of the SPCG method on the Beijing dataset [30]. The first contains (a) the original graph, (c) a sparse subgraph, and (e) the constraint part. (b) (d) (f) The corresponding matrices.

$\bar{\mathbf{x}} = \mathbf{R}_H^{-1}\mathbf{Q}_H^T\mathbf{b}_H$ of the subgraph part with the corresponding Gaussian log-likelihood $\|\mathbf{R}_H\mathbf{x} - \mathbf{c}_H\|^2$ where $\mathbf{c}_H = \mathbf{Q}_H^T\mathbf{b}_H$. Therefore, the original objective function becomes

$$f(\mathbf{x}) = \|\mathbf{R}_H\mathbf{x} - \mathbf{c}_H\|^2 + \|\mathbf{A}_C\mathbf{x} - \mathbf{b}_C\|^2 \tag{19}$$

Now we re-parametrize the problem in terms of the whitened deviation from the prior $\mathbf{y} = \mathbf{R}_H\mathbf{x} - \mathbf{c}_H = \mathbf{R}_H(\mathbf{x} - \bar{\mathbf{x}})$. By substituting $\mathbf{x} = \bar{\mathbf{x}} + \mathbf{R}_H^{-1}\mathbf{y}$ in (19), we obtain

$$\bar{f}(\mathbf{y}) = \|\mathbf{y}\|^2 + \|\mathbf{A}_C\mathbf{R}_H^{-1}\mathbf{y} - \mathbf{d}\|^2 \tag{20}$$

where $\mathbf{d} = \mathbf{b}_C - \mathbf{A}_C\bar{\mathbf{x}}$. Similarly, the matrix form of the new problem is

$$\begin{bmatrix} \mathbf{I} \\ \mathbf{A}_C\mathbf{R}_H^{-1} \end{bmatrix} \mathbf{y} = \begin{bmatrix} \mathbf{0} \\ \mathbf{d} \end{bmatrix} \tag{21}$$

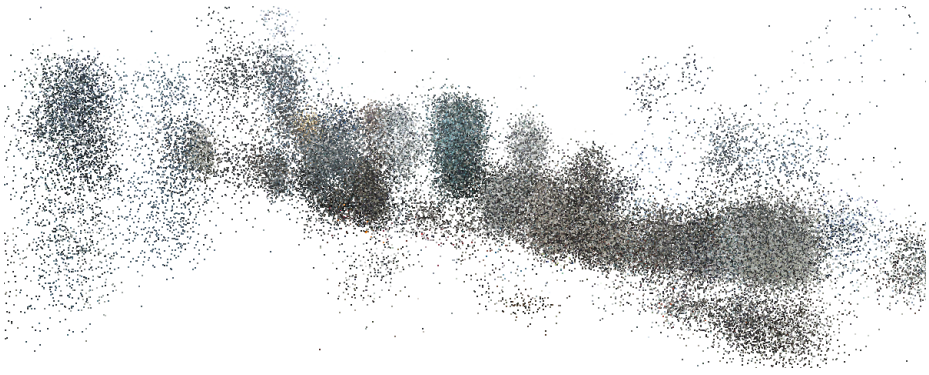
which can be solved by using the CGLS method. This strategy is more efficient than the first one because the matrix-vector multiplication on the $\mathbf{A}_H$ is not necessary.

In addition, I also visualize the solutions obtained from the subgraph and the solutions from the original graph in Figure 9. We can see that although the solution of the subgraph is blurry and hence inferior to that of the original graph, we can use it to build a preconditioner to solve the original graph efficiently.

Now let us examine the subgraph preconditioners in terms of Condition 1. Subgraph preconditioners satisfy the first condition because solving a sparse subgraph is always efficient, e.g. a spanning tree can be solved in linear time. Therefore the resulting preconditioner will be sparse and efficient to apply [26, 29]. Moreover, the quality of the subgraph preconditioners can be analyzed by support theory [15], which will be introduced in the next section.

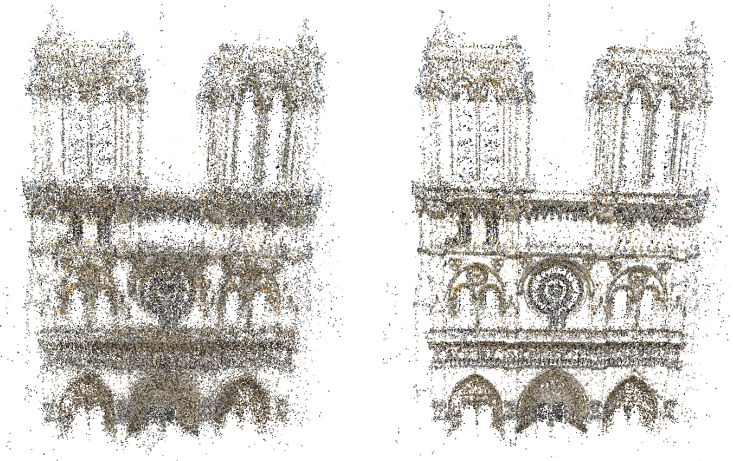
2.7 Support Theory

Although subgraph preconditioners demonstrates promising performance to solve the SLAM and SfM problems [30, 53], how to find a good subgraph is still an open



(a)

(b)



(c) (d)

Figure 9: The solutions obtained from solving (a) (c) a subgraph and (b) (d) the original graph of the *Chicago-2* dataset (from Grant Schindler) and the *NotreDame* datasets [95] respectively. Note the solutions of the subgraphs are more blurry than (inferior to) those of the original graphs, but they could serve as good preconditioners to solve the original graphs.

problem. To this end, I introduce support theory [15] to measure the quality of subgraph preconditioners. I will briefly summarize the support theory in this section.

2.7.1 Generalized Condition Number

The generalized condition number is a measure for the convergence speed of the preconditioned conjugate gradient method [87]. Namely, the generalized condition number for a pair of positive and definite matrices $\mathbf{M}_G$ and $\mathbf{M}_H$ is defined as

$$\kappa(\mathbf{M}_G, \mathbf{M}_H) = \frac{\lambda_{\max}(\mathbf{M}_G, \mathbf{M}_H)}{\lambda_{\min}(\mathbf{M}_G, \mathbf{M}_H)} \tag{22}$$

where $\lambda_{\max}(\mathbf{M}_G, \mathbf{M}_H)$ and $\lambda_{\min}(\mathbf{M}_G, \mathbf{M}_H)$ denote the largest and smallest generalized eigenvalue respectively, and the subscript H denotes a preconditioner. The generalized condition number is known to be inversely proportional to the worst-case convergence speed of PCG [87, 12]. In this dissertation, the roles of $\mathbf{M}_G$ and $\mathbf{M}_H$ are played by the outer product of the Jacobian matrix, i.e. $\mathbf{M}_G = \mathbf{A}_G^T \mathbf{A}_G$ and $\mathbf{M}_H = \mathbf{A}_H^T \mathbf{A}_H$. Hereafter I will refer to $\mathbf{M}_G$ as the original system matrix and $\mathbf{M}_H$ as the preconditioner system matrix.

2.7.2 Support Number

The generalized condition number measures the quality of a preconditioner with the ratio of extreme eigenvalues, but directly optimizing this measure is not trivial. Recently, support theory [15] has been proposed to assess the quality of preconditioners for symmetric and positive semidefinite linear systems. Here I provide a brief introduction to support theory. The readers may refer to [15] for details.

Central to support theory is the notion of *support number*:

Definition 2. *The support number of a pair of square matrices* $\mathbf{M}_G \in \mathbb{R}^{n \times n}$*, and* $\mathbf{M}_H \in \mathbb{R}^{n \times n}$ *is defined as*

$$\sigma(\mathbf{M}_G, \mathbf{M}_H) = \min\{t \in \mathbb{R} \mid \tau \mathbf{M}_H \succeq \mathbf{M}_G, \forall \tau \geq t\}. \tag{23}$$

In other words, the support number is the smallest number of "copies" that we need for $\mathbf{M}_H$ in order to dominate $\mathbf{M}_G$ in the Loewner sense, that is, for $\tau\mathbf{M}_H - \mathbf{M}_G$ to be positive semidefinite [108]. Another interpretation of the support number is that the shape of the quadratic function associated with $\tau\mathbf{M}_H - \mathbf{M}_G$ is convex.

In particular, the generalized condition number and the support number are connected via the following property:

Proposition 3. *Suppose* $\mathbf{M}_G \in \mathbb{R}^{n\times n}$ *and* $\mathbf{M}_H \in \mathbb{R}^{n\times n}$ *are symmetric and positive definite, then*

$$\kappa\left(\mathbf{M}_G, \mathbf{M}_H\right) = \sigma\left(\mathbf{M}_G, \mathbf{M}_H\right) \cdot \sigma\left(\mathbf{M}_H, \mathbf{M}_G\right). \tag{24}$$

This proposition suggests that $\mathbf{M}_H$ is a good preconditioner for $\mathbf{M}_G$ if both matrices can *support* each other with as little additional help as possible. Therefore we can instead focus on finding a preconditioner that minimizes the product of the two support numbers in (24).

2.7.3 Embedding Matrix

Now let us turn our discussion back to the Jacobian matrices ($\mathbf{A}_G$ and $\mathbf{A}_H$) and explain another important notion in support theory: the *embedding matrix*. An embedding matrix $\mathbf{W}$ contains the coefficients to linearly synthesize each row in matrix $\mathbf{A}_G$ by using the rows in matrix $\mathbf{A}_H$. This notion is useful to characterize the support number via the following theorem:

Theorem 4. *Suppose* $\mathbf{A}_G \in \mathbb{R}^{m\times n}$ *is in the range of* $\mathbf{A}_H \in \mathbb{R}^{p\times n}$, $\mathbf{M}_G = \mathbf{A}_G^T\mathbf{A}_G$, *and* $\mathbf{M}_H = \mathbf{A}_H^T\mathbf{A}_H$, *then*

$$\sigma\left(\mathbf{M}_G, \mathbf{M}_H\right) = \min_{\mathbf{W}} \|\mathbf{W}\|_2^2 \;\; \textit{subject to} \;\; \mathbf{W}\mathbf{A}_H = \mathbf{A}_G. \tag{25}$$

Theorem 4 shows that the squared spectral norm (largest singular value) of any embedding matrix provides an upper bound for the support number. The better

embedding matrix we identify, the lower upper bound for the support number we obtain. However, directly working with this metric could be inefficient because there is no closed-form expression to compute the spectral norm of a matrix.

Fortunately, there are simpler matrix functions that yield upper bounds for the spectral norm, and consequently for the support number. One of them is the Frobenius norm $\|\mathbf{W}\|_F$, which is defined as the square root of the sum of squared elements in the matrix. The consequence of this fact is well-known in numerical linear algebra, namely $\|\mathbf{W}\|_2^2 \leq \|\mathbf{W}\|_F^2$ [41]. The Frobenius norm is easier to work with as the rows of the embedding matrix can be considered independently.

2.7.4 Support Graph Theory

Bern et al. [11] used support theory to study this particular class of system matrices where each row of the matrix has only two nonzeros with the same magnitude but opposite signs. These system matrices can be transformed into a simple graph where each vertex denotes a variable, and each edge correspond to a row in the matrix with a weight defined by its magnitude. Note that the outer product of these matrices are called Laplacians and they have been studied in literature [24].

Under this definition, the embedding matrix defined in Theorem 4 has an intuitive combinatorial interpretation. The key idea is the notion of *path embedding*. Given a graph $G = (V, E, W)$, the path embedding of an edge $e = (u, v)$ in G refers to how to go between vertices u and v by using the edges in E. Similarly, given two graphs with an identical vertex set, the notion of *graph embedding* refers to a collection of path embeddings for the edges in one graph in the other graph.

To characterize the quality of a graph embedding, the notion of stretch is introduced. Suppose T is a spanning tree on V, then for every edge $e = (u, v) \in E$, there is a unique path in T connecting vertices u and v. The stretch of an edge $e \in E$ with

respect to the graph T is defined as

$$\mathbf{st}(T, e) = \sum_{e' \in \pi(T,e)} \frac{W(e)}{W(e')} \tag{26}$$

where $\pi(T, e)$ denotes the edges on the unique path between u and v in T. Note that the element $\frac{W(e)}{W(e')}$ fills in the embedding matrix, and the edges in $\pi(T, e)$ form the path embedding of the edge e in the graph T. Similarly, the stretch of graph G with respect to the graph T is defined as the sum of the stretches over all edges in E:

$$\mathbf{st}(T, G) = \sum_{e \in E} \mathbf{st}(T, e) \tag{27}$$

Intuitively speaking, the higher the stretch, the more time it takes for G to pass information with the graph T, negatively affecting the convergence. Boman and Hendrickson [13] showed that the stretch between a pair of graphs is actually equivalent to the Frobenius norm of an embedding matrix. It implies that we could derive good preconditioners by developing a graph algorithm that minimizes the stretch.

Another two combinatorial concepts derived from the graph embedding are called *congestion* and *dilation*. In a nutshell, the congestion denotes the level of the mostly loaded edge in the graph embedding. Likewise, the dilation indicates the maximum stretch of all path embeddings. Algebraically, they correspond to the matrix 1-norm and ∞-norm of embedding matrix respectively.

In sum, the support theory provides a tool to assess the quality of preconditioners for all positive definitite linear systems. When the system matrix is Laplacian, the problem of finding a good preconditioner can be transformed into a problem of finding a graph that minimizes the corresponding combinatorial quantities.

Note that the requirement on the outer product of the system matrix to be Laplacian can be relaxed to be symmetric and diagonally dominate matrix. This generalization allows the system matrix to have rows with only one nonzero element.

2.7.5 The Quality of Subgraph Preconditioners

Here I use support theory to analyze the subgraph preconditioners. By definition, if H is a spanning subgraph of G, i.e. $\mathbf{A}_H$ consists of a subset of the rows of $\mathbf{A}_G$, then by using Theorem 4 we know $\sigma(\mathbf{M}_H, \mathbf{M}_G) \leq 1$ because there exists an embedding matrix which is a proper subset of an identity matrix. This statement is true for all well-posed linear systems. Therefore we only need to analyze the other support number $\sigma(\mathbf{M}_G, \mathbf{M}_H)$.

As shown in support graph theory, the support number $\sigma(\mathbf{M}_G, \mathbf{M}_H)$ bears a graphical interpretation when the Jacobian matrix is an *oriented incidence matrix*, where each row has only two nonzeros with the same magnitude but opposite signs [11]. In this setting, the Jacobian matrices $\mathbf{A}_G$ and $\mathbf{A}_H$ can be transformed into weighted graphs G and H respectively. Boman and Hendrickson showed that the *stretch* between G and H is equivalent to the Frobenius norm of the embedding matrix, which is an upper bound of the support number $\sigma(\mathbf{M}_G, \mathbf{M}_H)$ [13]. Many results on finding a low-stretch spanning tree or graph sparsifiers have been published [96, 35, 65, 3].

However, the stretch cannot be directly used to evaluate the subgraph preconditioners for the SLAM and SfM problems because here the Jacobian matrices are more general than oriented incidence matrices: they are block-structured and each nonzero block could have arbitrary values. This limitation motivates us to develop a new measure for the quality of subgraph preconditioners for SLAM and SfM.

CHAPTER III

INTUITIONS ABOUT SUBGRAPH PRECONDITIONERS

3.1 Probabilistic Interpretation of Support Number

Support theory provides an *algebraic* tool to analyze preconditioners. Here my goal is to interpret support theory from a *probabilistic* point of view. This requires interpreting the support number in terms of the factor graph and the associated energy functions. Consider a factor graph expressed as $f(\theta) = \prod f_i(\theta_i)$ where θ denotes all variables, $f(\cdot)$ denotes the function that we want to maximize, and $\theta_i \subseteq \theta$ denotes the variables associated with the ith factor $f_i(\cdot)$. Suppose each factor can be written as a negative exponential function, i.e. $f_i(\theta_i) = \exp^{-e_i(\theta_i)}$, where $e_i(\theta_i)$ is a non-negative energy function associated with the ith factor, we can have the following equation

$$f(\theta) = \prod f_i(\theta_i) = \prod \exp^{-e_i(\theta_i)} = \exp^{-\Sigma e_i(\theta_i)} = \exp^{-E(\theta)} \tag{28}$$

where $E(\theta) = \Sigma e_i(\theta_i)$ is a summation of energy terms that we want to minimize.

To facilitate further discussion, I will use a subscript to denote the associated factor graph. For example, $f_G(\theta)$ denotes the original factor graph and $f_H(\theta)$ denotes the factor graph of a preconditioner. Now we can express the notion of *supporting* a factor graph G with another factor graph H by defining a scalar τ_0 as the smallest number such that $f_G(\theta) \geq (f_H(\theta))^\tau$ for any $\tau \geq \tau_0$ and for all θ. Expressing this equation in terms of the energy functions, we have

$$\tau E_H(\theta) \geq E_G(\theta) \quad \forall\theta, \forall\tau \geq \tau_0 \tag{29}$$

where τ_0 can be considered as the largest ratio between the two energy functions. Note that τ_0 is independent of the value of θ and only serves to compare the *functionals.*

Essential to our applications is a special case where each $f_i(\theta)$ is a Gaussian function. The corresponding energy can be expressed as a quadratic function $E_i(\theta) = \frac{1}{2}\theta^T\Lambda_i\theta$ where Λ_i is the information *matrix* of the ith factor. Since information is additive, the energy of the entire Gaussian factor graph can be written as

$$E(\theta) = \frac{1}{2}\theta^T\Lambda\theta \tag{30}$$

where Λ is the information matrix of the joint Gaussian distribution. By substituting (30) into (29), we obtain a Gaussian probabilistic interpretation of the support number

$$\tau\theta^T\Lambda_H\theta \geq \theta^T\Lambda_G\theta \quad \forall\theta, \forall\tau \geq \tau_0 \tag{31}$$

or equivalently

$$\tau\Lambda_H \succeq \Lambda_G, \quad \forall\tau \geq \tau_0. \tag{32}$$

From (32), we can see that τ_0 is exactly the support number between the information matrices of the original factor graph and the preconditioner. This observation is important, as it ties support theory and the idea of preconditioning back to the factor graph. We believe this is the first place to ever establish this connection.

3.2 Graphical Interpretation of Subgraph Preconditioners

The previous analysis affords an intuitive interpretation of the preconditioning (reparametrization) process with the *shape* of the distribution of the Gaussian factor graph. We start by representing a Gaussian factor graph distribution with the paraboloid associated with its quadratic energy function. Since the focus of preconditioning is exclusively to adjust the *curvature*, without loss of generality, we assume that any such paraboloid goes through the origin. If we cut the paraboloid perpendicularly on its axis of symmetry, we obtain an ellipsoidal section which uniquely represents the distribution. One distribution *dominates* the other if and only if its ellipsoid is entirely *inside* the other's ellipsoid (see Figure 10).

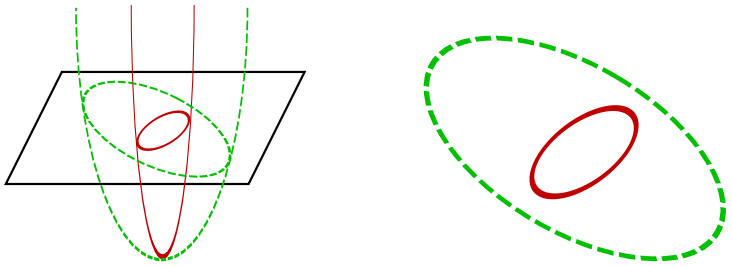


Figure 10: Illustration of the dominance relationship between two Gaussian distributions via the associated paraboloids and horizontal ellipsoid sections. (a) The green distribution is strictly larger than the red one at each point in space, which means that the green paraboloid is completely contained in the red paraboloid. (b) The cross-section of the two paraboloids with a horizontal hyperplane. The resulting green ellipsoid is entirely inside the red ellipsoid.

We can now illustrate the support number, previously expressed in terms of energy functions in (29), via a double inclusion of ellipsoids in the first column of Figure 11. Suppose we represent the distribution of the full factor graph G as a red ellipsoid, then for any subgraph H of G, the associated energy will always be smaller, therefore the associated ellipsoid (colored green) shall encompass the red ellipsoid. If we *scale up* the energy of H by a ratio of τ, we obtain a version of the green ellipsoid *scaled down* by a factor of τ (colored blue). As explained previously our objective is to find the smallest value of τ so that the blue ellipsoid dominates the red one.

With this convention, preconditioning has a natural and visual interpretation as follows. The effect of preconditioning is to linearly transform the subgraph distribution and any scaled version into a spherical ones. This means that the green and the blue ellipsoids will become spheres, having the same relative scaling factor as before. Since the re-parametrization is a linear transformation of the variable space, it will not affect the *inclusion* relationship among the three ellipsoids. That is, the re-parametrized red ellipsoid is still between the green and blue ellipsoids, and the shape of the new red ellipsoid characterizes the condition number of the re-parametrized system. The above interpretation is illustrated in the second column of Figure 11.

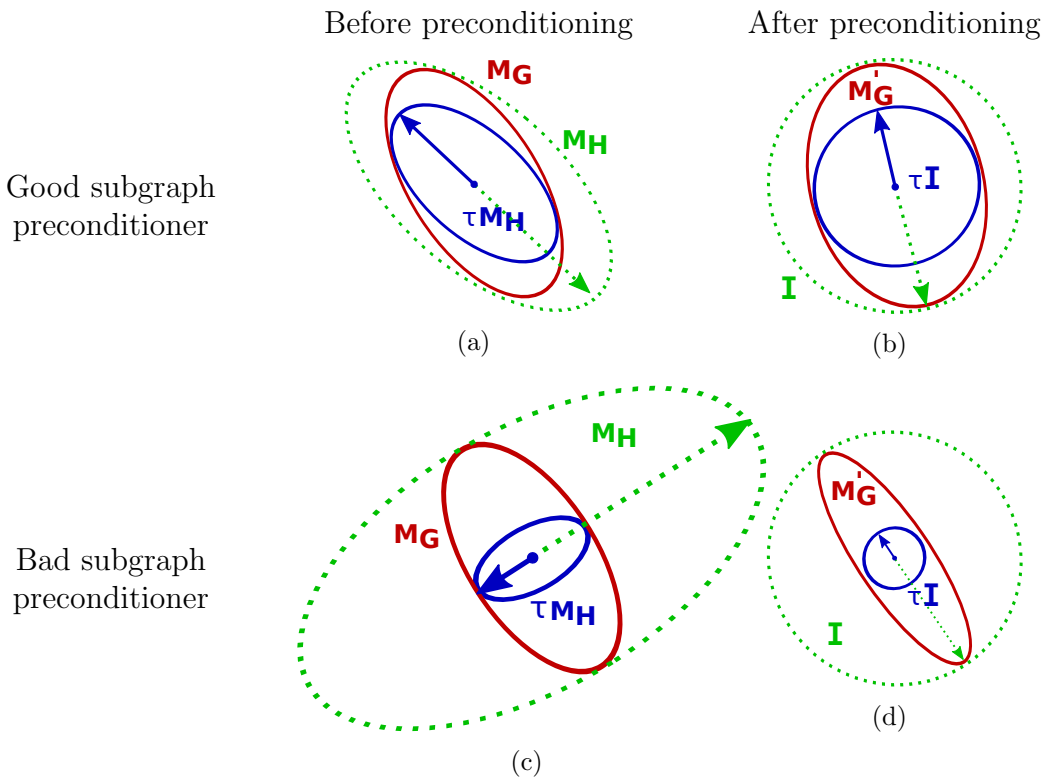


Figure 11: Illustration of different preconditioners in terms of the ellipse representation in Figure 10. The rows correspond to good and bad subgraph preconditioners respectively, while the left and right columns show the ellipse shapes before and after preconditioning. In each plot, the red ellipse denotes the energy function of the original graph, the green ellipse denotes that of the preconditioner. The dotted-line ellipse is scaled up (blue ellipse) to dominate the other one. The first row illustrates a good subgraph preconditioner where the red ellipse becomes more spherical, and hence the new problem becomes better-conditioned. The second row illustrates a bad subgraph preconditioner, in which the preconditioning fails its goal, due to a misalignment of the two functions.

CHAPTER IV

PERFORMANCE EVALUATION OF SUBGRAPH PRECONDITIONERS

Although the subgraph preconditioners have been applied to solve the SLAM problems [30], but how do subgraph preconditioners compare with the other preconditioners is still unclear. To this end, I conduct an empirical study to compare the performance of different preconditioning techniques on typical SLAM problems. The following preconditioners will be tested:

- the block Jacobi preconditioner [87],
- the multi-level incomplete QR (MIQR) preconditioner [74],
- a random spanning tree augmented with $c \cdot n$ additional edges (sptree+cn), where $c \geq 0$ is a parameter, and n is the number of robot poses.

The first two preconditioners serve as baseline, while the last one is used to characterize the empirical performance of the subgraph preconditioners. To build a random

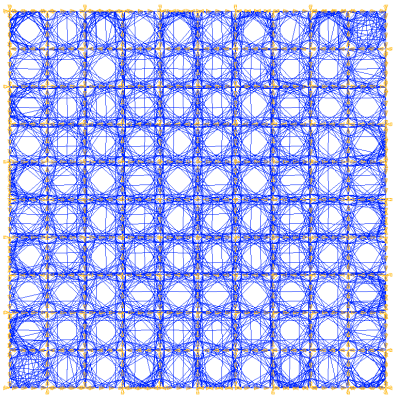


Figure 12: The bird's-eye view of a sample Blockworld problem with 1,000 robot poses (yellow) and 10,000 constraints (blue).

spanning tree preconditioner, I assign a random weight from 1 to 100 to each edge of the graph, and find the maximum-weighted spanning tree with Kruskal's algorithm [67]. Then I augment the tree with $c \cdot n$ additional random edges that were not in the spanning tree. Once the subgraph structure is determined, I use CHOLMOD to solve the subgraph to build the preconditioner.

To facilitate the comparison, I generated a synthetic SLAM problem that simulates a robot traversing a block world. Hereafter I will refer to them as the Blockworld problem (Figure 12).

For each instance of the problems. I attached a prior factor to the first robot pose to make the SLAM problems well-posed. In addition, for each robot pose I added twenty relative constraints to its closest neighbor poses, and these measurements are contaminated by zero-mean and normally distributed noise. The initial values of the robot poses are computed by adding normally distributed noise to the ground truth. Hereafter the above setting will be referred to as the *1-prior* scenario.

In addition, I tested another *n-prior* scenario where additional prior factors (L2 regularization terms) are attached to all robot poses. This scenario corresponds to the cases where GPS measurements are available or damped Newton's method (e.g. the Levenberg-Marquardt algorithm) is used to solve the full SLAM problem. In all of our experiments, I used the block diagonal of the outer product of the Jacobian matrix as the regularization terms.

I compared these preconditioners in three aspects: (1) the time to build the preconditioners, (2) the generalized condition numbers, (3) the time to solve linearized SLAM problems, (4) the spectrum of the preconditioned systems, and (5) the time to solve full SLAM problems.

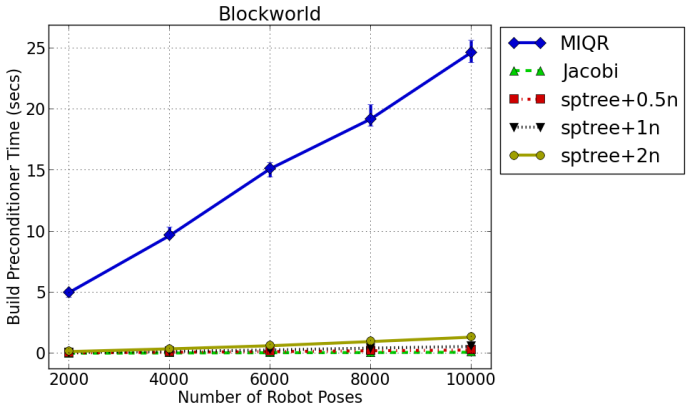


Figure 13: The time to build the preconditioners for linearized Blockworld problems.

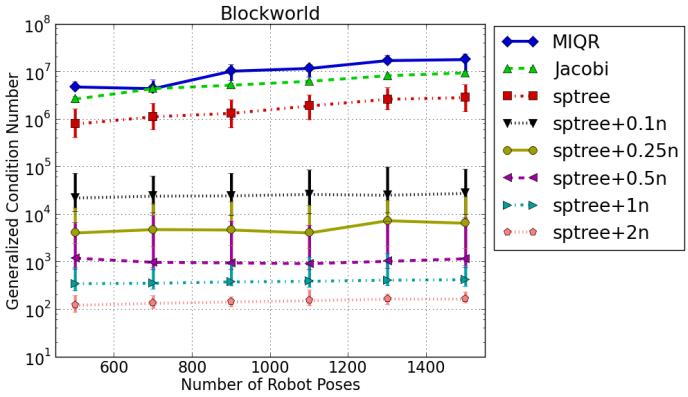


Figure 14: The generalized condition numbers of linearized 1-prior Blockworld problems. The figure shows the tenth percentile, the median and the ninetieth percentile.

4.1 Building Preconditioners

I compared the time to build the preconditioners for the Blockworld problems and show the results in Figure 13. We can see that MIQR took significant more time to build the preconditioners, which make it undesirable for the Blockworld problems. The Jacobi preconditioner is very efficient due to its simplicity. The time to build subgraph preconditioners is comparable to the Jacobi preconditioner, but the running time grows with the number of edges. It suggests that the subgraph should stay *sparse*, otherwise the cost of building the subgraph preconditioners will dominate.

4.2 Generalized Condition Numbers

I examined the generalized condition numbers of different preconditioners on 1-prior Blockworld problems. I solved each problem by using the Gauss-Newton algorithm [84] with CHOLMOD. The linearization points during the course of reaching the local minimum are used to generate a set of linearized SLAM problems as the datasets of this experiment.

The Jacobi and the MIQR preconditioners are constructed based on their definitions. For each of the augmented spanning tree preconditioners, I generated twenty random samples per linearization point. Once the graph structure is determined, I used CHOLMOD to compute the preconditioner and then used ARPACK [72], a popular solver for sparse eigenvalue problems, to compute the condition numbers.

Figure 14 shows the generalized condition numbers of the linearized SLAM problems. For clarity, I did not plot the condition numbers of the original systems in the same figure because they are much larger (up to 10^{12}). In general, the condition numbers of sptree+cn are consistently smaller than those of the other preconditioners. This implies that a random spanning subgraph suffices to be a good preconditioner for these two problems. Also, the generalized condition number can be further reduced by augmenting additional edges to the spanning tree because these additional edges facilitate the derivation of better embedding matrices.

Interestingly, although the spanning tree (sptree) preconditioner is consistently better than the Jacobi and the MIQR preconditioners, the condition numbers still grow with the number robot poses, which is undesirable for large problems. However, after I augment each random spanning tree with more and more edges, the condition numbers tend to remain constant with respect to the number of robot poses. This indicates that such classes of subgraph preconditioners are able to capture the problem structure with high probability.

I repeated the same experiments for the n-prior scenario with various magnitudes

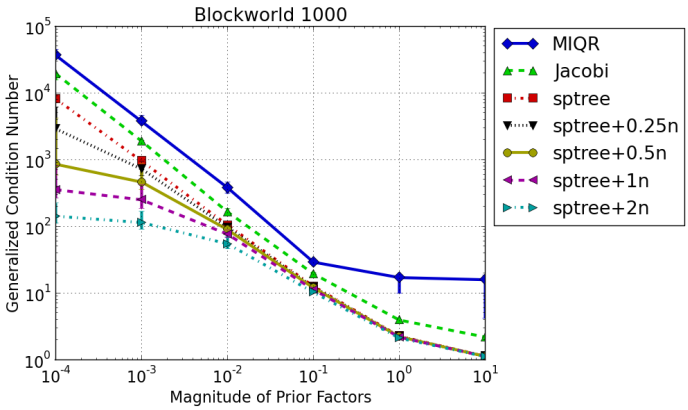


Figure 15: The generalized condition numbers of linearized n-prior Blockworld problems with 1000 robot poses. The figure shows the tenth percentile, the median and the ninetieth percentile.

of prior factors, and showed the results in Figure 15. We can see that when the magnitude increases, the original linear system becomes better-conditioned, and therefore the difference between preconditioners becomes less significant. This suggests that when the magnitude of regularization terms are large, even the simplest Jacobi preconditioner can provide good convergence speed for the *linearized* SLAM problems. Notice that although adding strong regularization terms will speedup the convergence of *linear* solvers, it will also slow down the convergence speed of the *nonlinear* solvers.

4.3 Solving Linearized SLAM Problems

The condition number measures worst-case convergence speed in terms of the iteration counts. To predict the actual runtime performance, we have to multiply the number of iterations by the cost of applying the preconditioner. Yet it is hard to combine all of these information into a single metric for analytical comparison. Therefore I compared the actual time and iterations required to solve the linearized Blockworld problems with the PCGLS method to converge up to two thresholds $\epsilon = 10^{-2}$ and $\epsilon = 10^{-6}$ (see Alg. 4 for the usage of ϵ). I used these thresholds to simulate the cases when PCGLS is used compute the *inexact* and *exact* Newton steps respectively.

I used PCGLS to solve linearized *1-prior* Blockworld problems and show the results

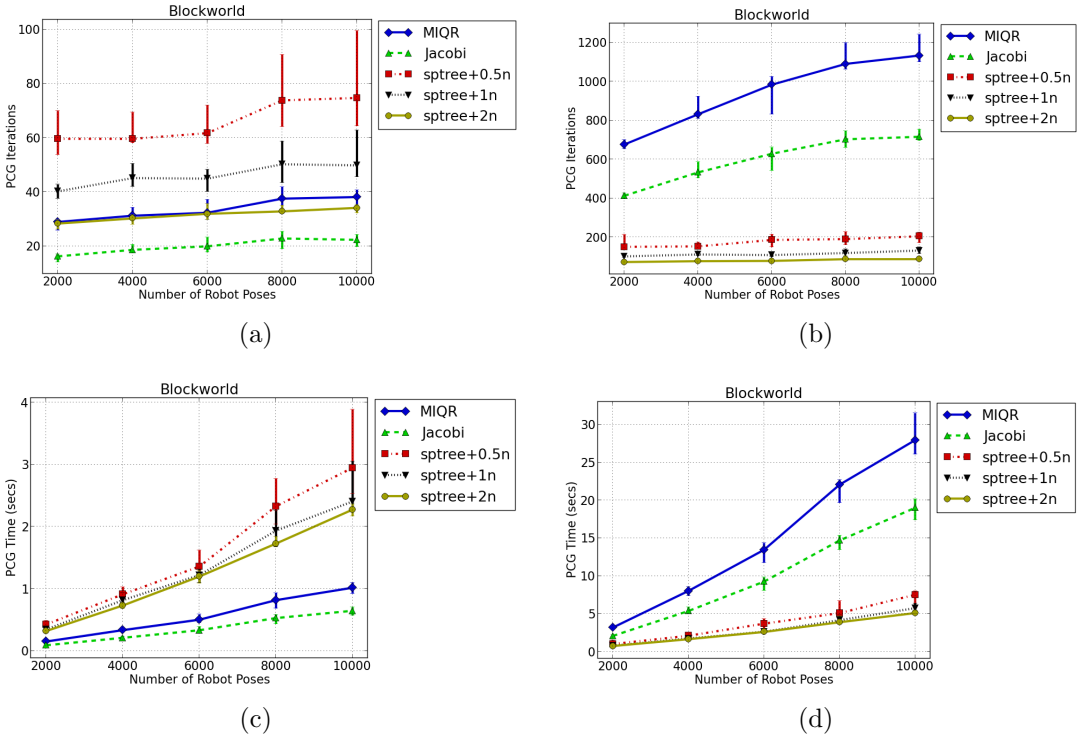


Figure 16: The performance of solving linearized 1-prior Blockworld problems. The first column corresponds to the stopping threshold $\epsilon = 10^{-2}$ while the second column corresponds to the stopping threshold $\epsilon = 10^{-6}$. The first row corresponds to the required CG iterations while the second row corresponds to the actual running time.

in Figure 16. We can see that Jacobi and MIQR preconditioners can converge to $\epsilon = 10^{-2}$ faster than the subgraph preconditioners. Yet subgraph preconditioners are significantly better than Jacobi and MIQR when more accurate solution is needed.

Notably, the generalized condition number does not predict the faster early convergence of Jacobi and MIQR, because it is a pessimistic estimation of the convergence speed. In the next section, I will further investigate this issue by comparing the spectrum of the preconditioned systems.

4.3.1 The Distributions of Generalized Eigenvalues

To further investigate the convergence properties of different preconditioners, I computed the distributions of generalized eigenvalues (spectrum) of a linearized Blockworld problem with roughly 1,000 poses, and plotted the results in Figure 17. We can

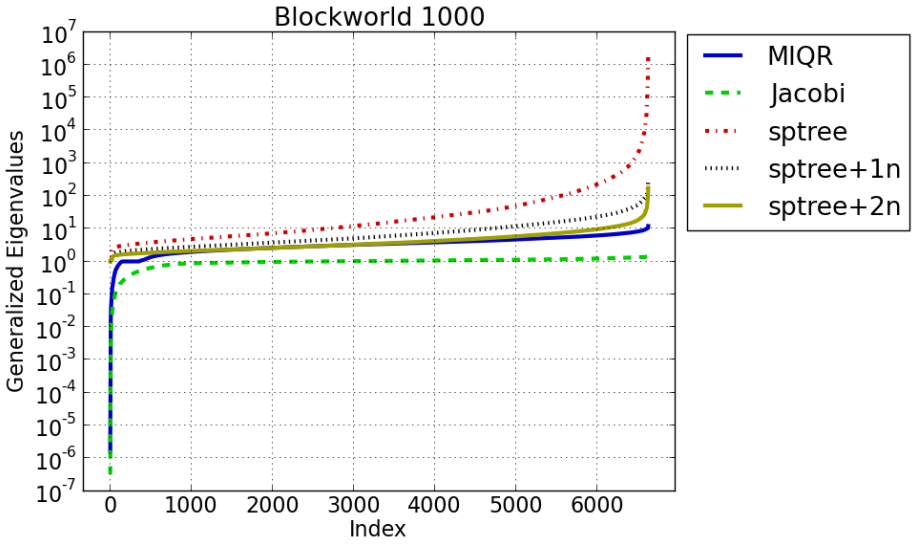


Figure 17: An example of the spectrum of a 1-prior Blockworld problems.

see that the spectrum of Jacobi has the following properties: (1) around 80% of the eigenvalues are clustered around 10^0, and (2) the rest eigenvalues have a much wider range (down to 10^{-7}). According to proposition 14, it implies that Jacobi preconditioner leads to fast convergence at the beginning because of the clustered eigenvalues, but may cause stagnant before finding highly-accurate solutions.

Subgraph preconditioners (sptree+cn) have spectrum different to the others preconditioners. First, the smallest eigenvalues are close to one, which confirms the prediction in Section 2.7.5. Second, the largest eigenvalues are inversely proportional to the complexity of subgraphs, which match the results in Section 4.2: the more edges the subgraph has, the better preconditioner it is. Third, around 90% of the eigenvalues are evenly distributed throughout its spectrum. This implies that the subgraph preconditioners will converge at a constant rate regardless of the magnitude of error. Fourth, the range of remaining eigenvalues is narrower than Jacobi, e.g., the largest eigenvalue of sptree+1n is only around 2×10^2. It implies subgraph preconditioners are unlikely to cause stagnant as Jacobi.

The spectrum of MIQR behaves like Jacobi preconditioner in the left end and

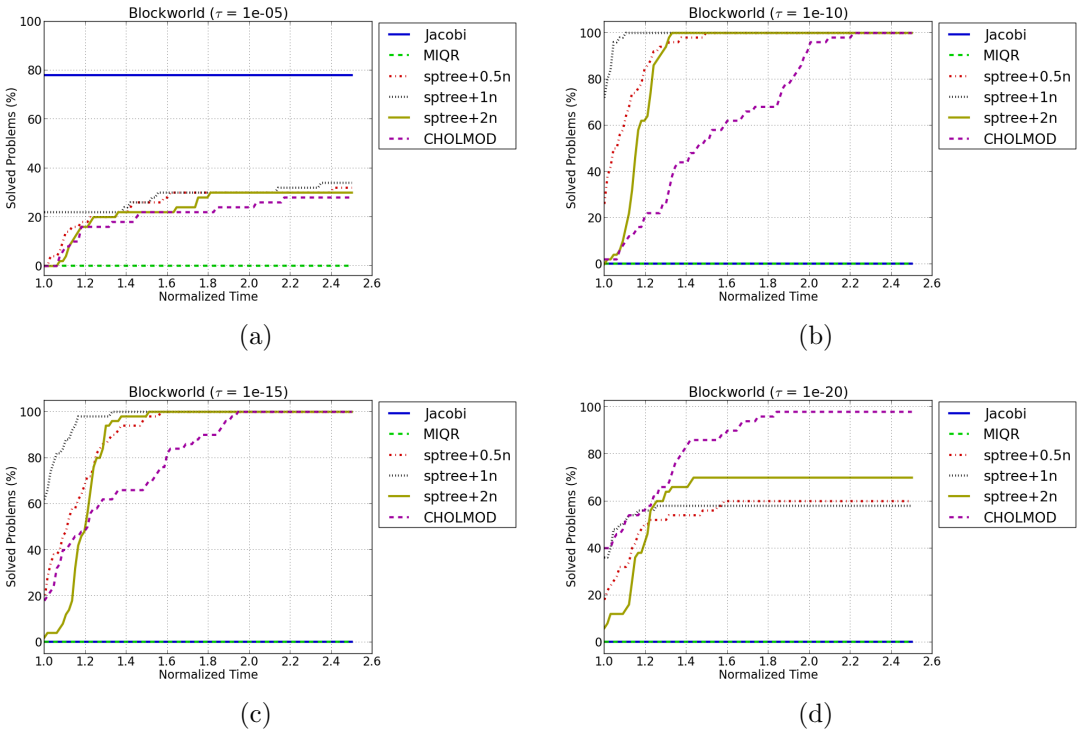


Figure 18: The performance profile of solving 1-prior Blockworld problems.

behaves like subgraph preconditioners in the middle and right end. This suggests that the convergence behavior of MIQR could be somewhat between Jacobi and sptree+cn.

I do not intend to reach a universal conclusion for the performance of these preconditioners on all SLAM scenarios, but it does suggest that combining different preconditioners could lead to a faster solver for linearized SLAM problems. For example, one could start with the Jacobi preconditioner to exploit the faster early convergence, and then continue with subgraph preconditioners to exploit the constant-rate convergence afterward. Note that Chen et al. also conducted experiments analyze the behavior of subgraph preconditioners for PDE problems, please refer to [22] for detail.

4.4 Solving Nonlinear SLAM Problems

I compared the runtime performance of solving full SLAM problems by using different preconditioners with the PCGLS method, and the state-of-the-art sparse direct solver (CHOLMOD) [23]. More specifically, I ran the Gauss-Newton algorithm to solve each

instance of the Blockworld problems. For the solvers based on PCGLS, I computed an *inexact* Newton step to update the current estimate by setting the stopping criteria as (1) the norm of the current gradient is smaller than 10^{-2} times of the initial gradient, or (2) the number of iterations exceeds five thousand (see Alg. 4 for detail). For CHOLMOD, it has to solve the linear systems exactly to compute the exact Newton step, and there is no freedom to combine CHOLMOD with *inexact* Newton methods.

I considered a problem solved if the error is τ-close to the optimum, i.e.

$$\frac{|e - e_*|}{|e_0 - e_*|} \leq \tau \tag{33}$$

where e_0 is the initial error, e is the current error, e_* is the minimum achievable error among all solvers, and τ is a threshold. I used the above settings to solve fifty Blockworld problems with 2,000 to 10,000 poses and report the execution times. Note that the time spent to linearize the problems is excluded to emphasize the difference between the solvers. In practice, linearizing the problems takes around 15% of the total execution time.

Showing all of the execution times or the convergence plots will be overwhelming and uninformative. Instead I summarized the results by using the recently-developed *performance profile* [31]. The main idea is to compare the solvers by the ratio of one solver's runtime to the best runtime among all solvers, with the solvers rated by the probability of solving the problems.

Figure 18 shows the performance profiles of solving full 1-prior SLAM problems under four thresholds. In each plot, the horizontal axis indicates the normalized time, while the vertical axis denotes the probability that a problem is solved. By definition, the best solver is the one that can solve the most problems in the least amount of time, and hence the corresponding curve should be as close to the upper left part of the plot as possible. We can also compare the solvers in terms their speed or their ability to solve a problem. For example, any vertical slice indicates the probability of solving a problem given a fixed amount of time. Likewise, any horizontal slice

indicates the required amount of time to solve a certain percentage of the problems.

I made the following observations to the results in Figure 18. (1) MIQR does not show as a winner in all plots due to its inefficiency of building the preconditioners. (2) Jacobi can reduce the error faster than the others at the early stage (Figure 18(a)), but fails to further reduce the error to a smaller threshold. This phenomenon confirms our results in Section 4.3. (3) CHOLMOD is slower than the iterative solvers unless a highly accurate solution is needed (Figure 18(d)). In practice, such a high quality solution is not always needed, and its memory requirement makes it infeasible for very large problems. (4) Subgraph preconditioners are the winners in Figures 18(b) and 18(c). Among all of the preconditioners, the sptree+1n is the best as it is 50-100% faster than CHOLMOD. I noticed that the sptree+0.5n and the sptree+2n are not as efficient because the former does not reduce the condition number sufficiently, while the latter, despite the fact that adding more edges reduces the condition number, becomes more expensive to apply in PCGLS, which degrades the overall performance. This phenomenon suggests that having an effective and compact subgraph is very important. Using too sparse subgraphs may not lead to effective preconditioners, while using too dense subgraphs may degrade the overall performance.

4.5 Summary

I applied Jacobi, MIQR preconditioners and (random) subgraph preconditioners to solve a simulated Blockworld problem. I compared them in terms of (1) the time to build the preconditioners, (2) the generalized condition numbers, (3) the efficiency of solving linearized SLAM problems, (4) the distribution of generalized eigenvalues, and (5) the efficiency of solving full SLAM problems.

I found that the best preconditioner is problem dependent. When the problem is inherently well-conditioned (e.g. n-prior scenario), or strong regularization terms are required to stabilize the nonlinear problem, the difference of preconditioners becomes

less significant. In this case, standard preconditioning techniques like the Jacobi preconditioner would work well. Yet when the problem is inherently ill-conditioned (e.g. 1-prior scenario), the subgraph preconditioners are better choices because they deliver more stable convergence behavior. The Jacobi preconditioner can reduce the error faster at the early stage but cause stagnant afterward.

I compared the performance of subgraph preconditioners with different complexity. I found that using a subgraph with appropriate complexity is crucial to the performance. Using too sparse subgraphs may not lead to effective preconditioners, while using too dense subgraphs may degrade the overall performance.

MIQR does not show as a winner in any cases. The main reason is its inefficiency in building the preconditioner for Blockworld problem. Regardless of this issue, its convergence behavior is also not superior to Jacobi and subgraph preconditioners.

I also combined these preconditioners with PCGLS and *inexact* Gauss-Newton method to solve *nonlinear* SLAM problems, and compared their performance with state-of-the-art sparse direct solver (CHOLMOD). My results showed that using subgraph preconditioners with PCGLS are better than CHOLMOD, unless a highly-accurate solution is needed.

Given a better understanding to the convergence behavior of these solvers, it suggests that there could be a better strategy to solve the problems by combining different linear solvers. One possibility is to use Jacobi at the early stage, then switch to subgraph preconditioners in the middle stage, and then switch to sparse direct method in the final stage.

CHAPTER V

SUPPORT-THEORETIC SUBGRAPH PRECONDITIONERS

In this chapter, I aim to build good subgraph preconditioners in the sense of generalized condition numbers. To this end, I use support theory [15] to derive a new metric for the quality of a *spanning tree* preconditioner. Then I use this metric to develop an algorithm based on Markov Chain Monte Carlo (MCMC) methods [40] to find a good spanning tree preconditioner, and then augmented it with additional edges to build a good subgraph preconditioner [106]. In addition, I apply heuristics and domain knowledge in SLAM to improve the efficiency of finding good subgraph preconditioners. I use these new subgraph preconditioners with the PCGLS method to solve large-scale SLAM problems and obtain promising results. The preliminary work of this chapter has been published in [52].

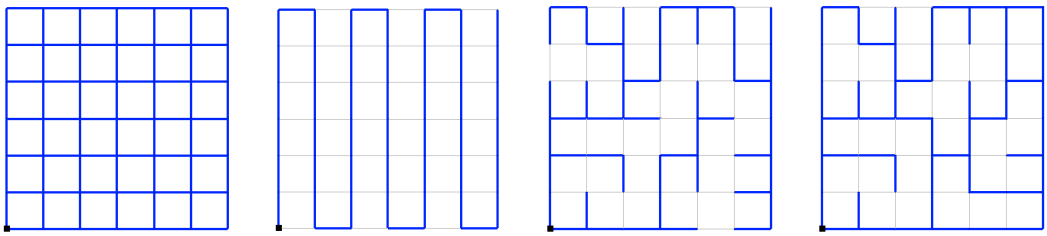


Figure 19: Illustration of the proposed algorithm with a simple grid graph. (a) The original graph. (b) The robot's trajectory as an initial spanning tree. (c) The spanning tree after thirty iterations of our algorithm. (d) A subgraph is built by inserting additional high-stretch edges to the spanning tree.

5.1 Generalized Stretch (GST)

In this section, I will define the notion of *generalized stretch* for the Jacobian matrices with the following properties: (1) the matrices are block-structured, (2) every nonzero block is invertible, (3) there is exactly one block-row with exactly one nonzero block, (4) the other block-rows have exactly two nonzero blocks, and (5) the matrix has full column rank. In SLAM, this setting resembles a scenario in which the robot knows its initial pose (a unary prior factor) in the world coordinate, and has sensors (e.g., odometry, loop-closure) to induce pose constraints (binary factors). Hereafter I will refer to a matrix satisfying these properties as an **A**-matrix. Since I exclusively work with block-structured matrices, the word "block" will be omitted for simplicity.

5.1.1 Canonical Form of an A-Matrix

To facilitate the exposition, I define the canonical form of an **A**-matrix as

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_0 \\ \mathbf{A}_1 \\ \vdots \\ \mathbf{A}_m \end{bmatrix} \tag{34}$$

where

$$\mathbf{A}_0 = \begin{bmatrix} \mathbf{A}_{0,0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{35}$$

is the row with one nonzero block, and

$$\mathbf{A}_i = \begin{bmatrix} \cdots & \mathbf{A}_{i,a_i} & \cdots & \mathbf{A}_{i,b_i} & \cdots \end{bmatrix} \tag{36}$$

is the ith row vector with two nonzero blocks (indexed by a_i and b_i), m is the number of binary factors, and n is the number of block variables. Every **A**-matrix can be transformed to this canonical form by permuting the rows and columns. An **A**-matrix

is indexed by the block variables, and therefore I define

$$\mathbf{A}(i,j) = \begin{cases} \mathbf{A}_{0,0} & \text{if } i = 0 \text{ and } j = 0, \\ \mathbf{A}_{i,j} & \text{if } 1 \leq i \leq m \text{ and } j \in \{a_i, b_i\}, \\ \mathbf{0} & \text{otherwise.} \end{cases} \tag{37}$$

5.1.2 Transformation to an A-Graph

Here I show how to transform an **A**-matrix into an **A**-graph. I define a graph $G = (V, E)$ where $V = \{v_1, v_2, \cdots, v_n\}$ denotes a set of vertices, each of which corresponds to a column in **A**, and $E = \{e_0, e_1, \cdots, e_m\}$ are the edges of G, each of which corresponds to a row in **A**. With slight abuse of notation, I define $\mathbf{A}(e_i) = \mathbf{A}_i$ that associates an edge to a block row, and $\mathbf{A}(e_i, v_j) = \mathbf{A}(i, j)$ that associates a pair of vertex v_j and edge e_i to a square block matrix.

An edge e_i is incident to the vertex v_j if $\mathbf{A}(e_i, v_j) \neq \mathbf{0}$. Moreover, two edges e_i and e_j are adjacent if they share a vertex, denoted as $e_i \cap e_j$. For a pair of adjacent edges e_i and e_j, I define a function

$$r_{\mathbf{A}}(e_i, e_j) = \mathbf{A}(e_i, e_i \cap e_j) \cdot \mathbf{A}(e_j, e_i \cap e_j)^{-1}, \tag{38}$$

which is a square matrix that represents the *ratio* between two edges with respect to the shared vertex.

I also define a path $P = (e_1\, u_1\, e_2\, u_2\, \cdots\, u_d\, e_{d+1})$ as a sequence of vertices and edges so that (1) it always starts and ends with an edge, and (2) the vertices and the edges always interlace with each other. The length of a path $|P|$ is defined as the number of vertices d. I also define $\tilde{P} = (e_2\, u_2\, \cdots\, u_d\, e_{d+1})$ as a sub-path of P by removing the first edge and vertex.

5.1.3 Path Embedding in a Spanning Tree

Central to our derivation is the notion of *path embedding*. Here I show how to compute the *path embedding* for any edge with respect to a spanning tree. I choose to

investigate this case because the path embedding in a spanning tree is unique and can be derived analytically.

More specifically, suppose $T = (V, E_T)$ is a spanning tree of G, the path embedding for an edge $e_s \in E$ with respect to T consists of a set of weights

$$\mathbf{w}_T(e_s) = \{\mathbf{w}_T(e_s, e) \mid \forall e \in E_T\} \tag{39}$$

so that e_s can be perfectly reconstructed, that is,

$$\sum_{e \in E_T} \mathbf{w}_T(e_s, e)\mathbf{A}(e) = \mathbf{A}(e_s). \tag{40}$$

Since T is a spanning tree, the weights are unique and can be derived analytically. Suppose $e_s = (v_a, v_b)$ is an edge to be embedded, there are two cases: (1) If $e_s \in E_T$, then the weights are all zeros except that $\mathbf{w}_T(e_s, e_s)$ is an identity matrix. (2) If $e_s \notin E_T$, then the weights can be derived by performing Gaussian elimination from the end vertices, v_a and v_b, to the root vertex v_r of T, which is defined as the vertex with the unary prior factor. Note that $v_r = v_1$ in our canonical representation.

After a series of algebraic calculations, we can derive the weights with respect to the ratio function defined in (38):

$$\mathbf{w}_T(e_s, e) = \begin{cases} \mathbf{0} & \text{if } e \notin P_T(v_a) \cup P_T(v_b) \\ r_{\mathbf{A}}(e_s, e) & \text{if } e \cap e_s = v \\ -\sum_{e' \in \mathbf{D}_T(e)} \mathbf{w}_T(e_s, e') \cdot r_{\mathbf{A}}(e', e) & \text{o/w,} \end{cases} \tag{41}$$

where $P_T(v)$ is defined as e_0 plus the edges on the unique path between v and the v_r in T, $\mathbf{D}_T(e)$ denotes a set of edges incident to e in T leading to the vertices of greater depth. The depth of a vertex is defined as its distance to the root vertex.

5.1.4 Generalized Stretch

In support theory, the *stretch* of an edge is defined as the squared Frobenius norm of the path embedding for the oriented incidence matrices [102]. Here I use (41) to

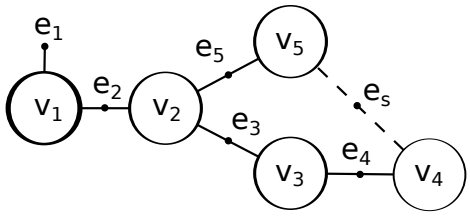


Figure 20: A simple example to illustrate the generalized path embedding.

define the notion of *generalized stretch* for the **A**-matrices:

$$\mathbf{gst}_T(e_s) = \sum_{e \in E_T} \|\mathbf{w}_T(e_s, e)\|_{\mathrm{F}}^2. \tag{42}$$

It has been shown in [13] that the sum of stretches over all edges in G, namely

$$\mathbf{gst}_T(G) = \sum_{e_s \in E_G} \mathbf{gst}_T(e_s), \tag{43}$$

measures how well a spanning tree T serves as a preconditioner for a graph G because it corresponds to an upper bound of the generalized condition number. I will use this property to derive good subgraph preconditioners.

5.1.5 Example

Here I provide an example to explain the idea of path embedding in **A**-graph. I start by investigating a simple example in Figure 20 where we have five variables and a spanning tree T (solid-edge), and one edge to be supported (dashed-edge).

Given the **A**-matrix of the spanning tree and the dashed-edge, our goal is to analytically derive the edge weights $\mathbf{w_T}(e_s, \cdot)$ to satisfy the following equation:

$$\begin{bmatrix} \mathbf{w}_1 & \mathbf{w}_2 & \mathbf{w}_3 & \mathbf{w}_4 & \mathbf{w}_5 \end{bmatrix} \begin{bmatrix} \mathbf{H}_{11} & & & & \\ \mathbf{H}_{21} & \mathbf{H}_{22} & & & \\ & \mathbf{H}_{32} & \mathbf{H}_{33} & & \\ & & \mathbf{H}_{43} & \mathbf{H}_{44} & \\ & \mathbf{H}_{52} & & & \mathbf{H}_{55} \end{bmatrix} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{G}_4 & \mathbf{G}_5 \end{bmatrix} \tag{44}$$

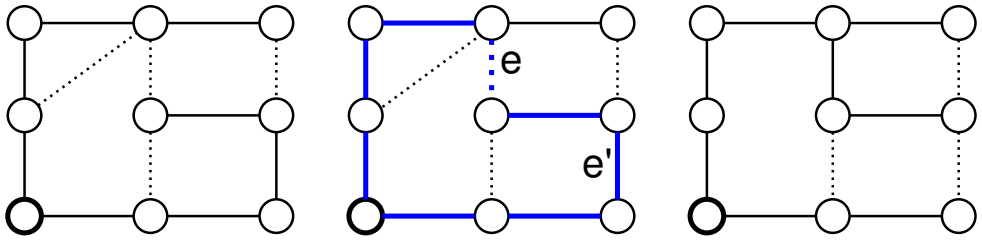


Figure 21: Illustration of one iteration of our algorithm. (a) The current spanning tree T (solid edges). (b) Suppose the off-tree edge e is sampled. Inserting e into T would induce the blue cycle. Suppose the edge e' is sampled from the cycle. (c) Swapping e and e' leads to a new tree T'.

where $\mathbf{w}_i := \mathbf{w}_{\mathbf{T}}(e_s, e_i)$, and $\mathbf{H}_{ij}, \mathbf{G}_j$ are block matrices. According to (38), we can define the ratio matrices as

$$r(e_2, e_1) = \mathbf{H}_{21}\mathbf{H}_{11}^{-1},\ r(e_3, e_2) = \mathbf{H}_{32}\mathbf{H}_{22}^{-1},\ r(e_5, e_2) = \mathbf{H}_{52}\mathbf{H}_{22}^{-1},$$
$$r(e_4, e_3) = \mathbf{H}_{43}\mathbf{H}_{33}^{-1},\ r(e_s, e_4) = \mathbf{G}_4\mathbf{H}_{44}^{-1},\ r(e_s, e_5) = \mathbf{G}_5\mathbf{H}_{55}^{-1}.$$

By a series of algebraic calculation, we can derive the path embedding as a function of the ratio matrices:

$$\mathbf{w}_5 = r(e_s, e_5), \qquad \mathbf{w}_4 = r(e_s, e_4), \qquad \mathbf{w}_3 = -r(e_s, e_4) \cdot r(e_4, e_3),$$
$$\mathbf{w}_2 = r(e_s, e_4) \cdot r(e_4, e_3) \cdot r(e_3, e_2) - r(e_s, e_5) \cdot r(e_5, e_2), \text{and}$$
$$\mathbf{w}_1 = -(r(e_s, e_4) \cdot r(e_4, e_3) \cdot r(e_3, e_2) \cdot r(e_2, e_1) - r(e_s, e_5) \cdot r(e_5, e_2) \cdot r(e_2, e_1)).$$

One can verify that the results match with the equations in (41).

5.2 Support-Theoretic Subgraph Preconditioners (STSP)

To find a good subgraph preconditioner, a common practice is to find a low-stretch spanning tree as the skeleton, and then augment it with high-stretch edges to further reduce the total stretch [96, 65]. I will follow the same principle to construct support-theoretic subgraph preconditioners.

5.2.1 Support-Theoretic Spanning Tree Preconditioner (STST)

I first consider the following problem

$$\min_{T} \ \mathbf{gst}_T(G), \tag{45}$$

but solving (45) is an NP-hard problem [75]. Instead I derive an algorithm based on MCMC techniques [40] to find a low-stretch spanning tree. The algorithm assumes an initial spanning tree T is available. For each iteration, I sample an edge $e \notin E_T$ with a probability proportional to $\mathbf{gst}_T(e)$. Inserting e into the spanning tree leads to a new subgraph $T^+ = (V, E_T \bigcup e)$, which contains an induced cycle $C_T(e)$. To obtain a spanning tree again, I pick an edge $e' \in C_T(e)$ uniformly at random, and swap e and e' to build a new spanning tree T'. If the new total stretch $\mathbf{gst}_{T'}(G)$ is smaller than the original total stretch $\mathbf{gst}_T(G)$, then I accept T' unconditionally. Otherwise, I accept T' with a probability following an exponential distribution of the logarithm of the ratio between two stretches. Thus the algorithm can be thought of as a Markov Chain based on Metropolis updates. The above procedure is illustrated in Figure 21. I repeat this procedure until convergence. In the end, I output the best spanning tree during the course.

5.2.2 Subgraph Construction

Given the best spanning tree T_* computed in the previous step, I construct a subgraph by inserting the edges with high stretch into the spanning tree. The rationale behind picking these edges is that they are likely to reduce the generalized condition numbers the most. I have examined two edge selection strategies. The first is to greedily pick the edges with the largest stretch. The second is to sample the edges with a probability according to their stretch. Please refer to Section 5.3.3 for the results. The key steps of the proposed algorithm are summarized in Algorithm 1.

Algorithm 1: The proposed algorithm

Input: G is the graph, T_0 is a spanning tree of G
Initialization: $s_0 = \mathbf{gst}_{T_0}(G)$
for $i = 0$ **to** maximum iterations **do**
  if *convergent* **then** break
  1. sample an edge $e \in G$ with probability $\propto \mathbf{gst}_{T_i}(e)$
  2. let $C_{T_i}(e)$ be the unique cycle in $T^+ = (V, E_T \bigcup e)$
  3. uniformly at random sample an edge e' from $C_{T_i}(e)$
  4. swap e and e' so that $T_i' = (V, E_T \bigcup e \backslash e')$
  5. compute $s_i' = \mathbf{gst}_{T_i'}(G)$
  if $s_i' < s_i$ **then**
    $T_{i+1} = T_i'$; $s_{i+1} = s_i'$
  else
    $x = \log(\frac{s_i'}{s_i})$
    $\alpha = \min(1, \lambda \exp(-\lambda x))$
    generate a random number $q \sim U[0, 1]$
    if $q \leq \alpha$ **then** $T_{i+1} = T_i'$; $s_{i+1} = s_i'$
    else $T_{i+1} = T_i$; $s_{i+1} = s_i$
  end
end
let $T_* = \text{argmin}_{T_i} \mathbf{gst}_{T_i}(G)$
augment T_* with edges (see text), and output the subgraph

5.2.3 Computational Complexity

Here I summarize the complexity of the each step of the algorithm. In the initialization step, computing $\mathbf{gst}_{T_0}(G)$ takes $O(md)$ where m is the number of off-tree edges and d is the average depth of the end vertices of the off-tree edges. In a balanced tree, d is close to $\log(n)$ where n is the number of vertices. In the inner loop of the algorithm, steps one to four can be done in $O(m)$. The fifth step can be done in $O(md)$ if we recompute it from scratch. Yet it can be improved by just recomputing the generalized stretches of the edges associated to the subtree of the edge e'.

5.3 Results

I conducted five experiments to evaluate the proposed algorithm: (1) I evaluated the efficiency of our MCMC algorithm. (2) I evaluated the quality of different spanning tree preconditioners. (3) Given a low-stretch spanning tree, I evaluated two different

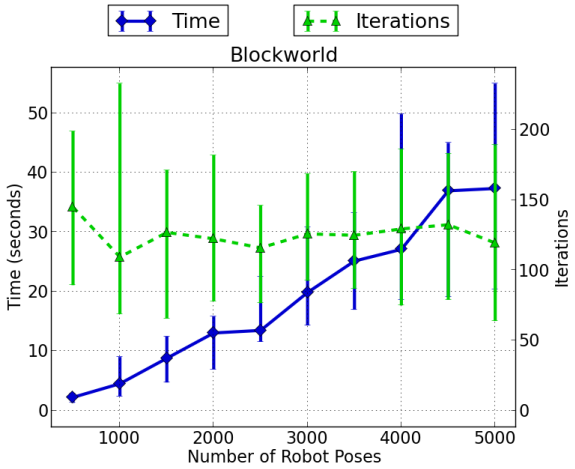


Figure 22: The efficiency of Algorithm 1.

edge selection strategies to construct a subgraph. (4) I evaluated the quality of different subgraph preconditioners. (5) I used these subgraph preconditioners with the PCGLS method to solve both synthetic and real SLAM problems, and compare the running time against the state-of-the-art sparse direct solver [23].

To facilitate the comparison, I generated a synthetic Blockworld SLAM problem, simulating a robot traversing a block world. The bird's-eye view of this problem is illustrated in Figure 12. For each instance of the Blockworld problem, I attached a prior factor to the first robot pose to make the SLAM problems well-posed. In addition, for each robot pose I added twenty relative constraints to its closest neighbor poses, and these measurements are contaminated by zero-mean and normally distributed noise. The initial values of the robot poses are computed by composing the measurements along the robot's trajectory.

For Algorithm 1, I set $\lambda = 10^3$ and considered the algorithm is convergent if the average decrease of relative stretch in the past 100 iterations is smaller than 10^{-3}. I ran all of the experiments on a PC with an Intel Core i7 CPU, and reported the tenth percentile, the median and the ninetieth percentile over fifty trials.

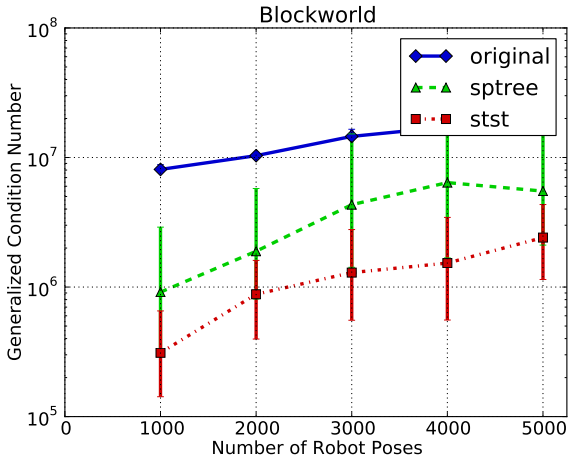


Figure 23: The generalized condition numbers of different spanning trees.

5.3.1 The Efficiency of Our MCMC Algorithm

I evaluated the efficiency of our MCMC algorithm by measuring the required time and iterations to converge for the Blockworld problem. For each instance of the Blockworld problem, starting from a random spanning tree, I applied our algorithm to find a low-stretch spanning tree and reported the results in Figure 22. we can see that as the problem size increases, the number of required iterations stays almost constant, which indicates that a good tree can be found in a constant number of edge swaps. However, the required time increases linearly with the problem size, which negatively affects the performance of our algorithm for large-scale problems.

5.3.2 Generalized Condition Numbers of Spanning Trees

I compared two spanning tree preconditioners for the Blockworld problem: (1) a random spanning of the entire graph (sptree), and (2) the support-theoretic spanning tree (stst) computed by the proposed algorithm. The first two settings characterize the empirical performance of an ad-hoc spanning tree. To build a random spanning tree, I assigned a random weight from 1 to 100 to each edge of the graph, and computed the maximum-weighted spanning tree with Kruskal's algorithm [67].

Once the spanning tree is determined, I used CHOLMOD [23], an efficient sparse

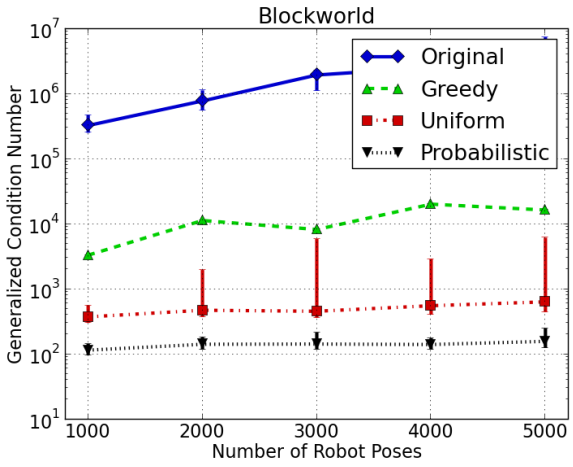


Figure 24: The generalized condition numbers resulted by using different edge selection strategies to build a subgraph.

direct solver, to compute the preconditioner, and used ARPACK [72] to compute the generalized condition numbers. I repeated this procedure for fifty times and reported the tenth percentile, the median and the ninetieth percentile in Figure 23. We can see that stst is significantly better than the other two approaches, and the results confirms that our algorithm indeed produces better spanning trees. However, the generalized condition numbers increase with the problem size which indicates that using a spanning tree preconditioner is not scalable.

5.3.3 Subgraph Construction

Given the low-stretch spanning tree computed in the previous step, I evaluated the performance of the three edge selection strategies to construct a subgraph. More specifically, I examined the following three strategies: (1) greedily pick the edges with the largest stretch (Greedy), (2) uniformly at random sample edges (Uniform), and (3) probabilistic sample the edges according to their stretch (Probabilistic).

I conducted experiments on the Blockworld problems. For each instance of the problems, I applied Algorithm 1 to computed a low-stretch spanning tree which serves as a baseline (Original). Then I used these strategies to insert n edges to each of the spanning trees to build subgraphs, where n is the number of robot poses. For the

Greedy setting, I sorted edges according to their stretch, and pick the top-n edges. For the Probabilistic setting, I sampled n additional edges with probability proportional to their stretch, and inserted them into the spanning tree to build a subgraph. Once the subgraph is determined, I used the same procedure described in Section 5.3.2 to compute the generalized condition numbers.

From the results in Figure 24, we can see that these subgraph preconditioners can improve the generalized condition numbers up to four orders of magnitudes. I also observed that the slopes of growth are flatter than those in the spanning tree experiments. It implies that inserting additional edges to a spanning tree indeed leads to a better and more scalable preconditioner. Comparing these three strategies, we can see that Greedy is worse than the other two strategies. I conjecture that it is because the edges chosen by Greedy may concentrate at a certain part of the graph, and therefore fail to reduce the stretch for the other parts of the graph. On the other hand, the edges chosen by the Uniform and Probabilistic strategies have a higher chance to spread over the graph, and therefore could reduce the total stretch even further. The Uniform strategy is better than Greedy up to an order of magnitude, but its performance is unstable due its larger variance. Finally, the probabilistic has the best and stable performance. Therefore, I used the Probabilistic strategy in the following experiments.

5.3.4 Generalized Condition Numbers of Subgraphs

I compared the generalized condition numbers of two subgraph preconditioners: (1) sptree augmented with random edges (sptree+cn), and (2) stst augmented with additional edges sampled by using the Probabilistic strategy (stst+cn), where c is a ratio of augmented edges and n is the number of robot poses.

From the results in Figure 25, we can see that (1) as the ratio of augmented edges c becomes larger, the generalized condition numbers of all subgraph preconditioners

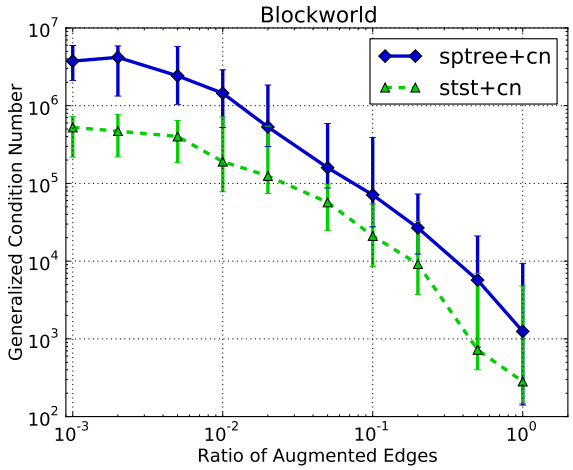


Figure 25: The generalized condition numbers of the subgraph preconditioners.

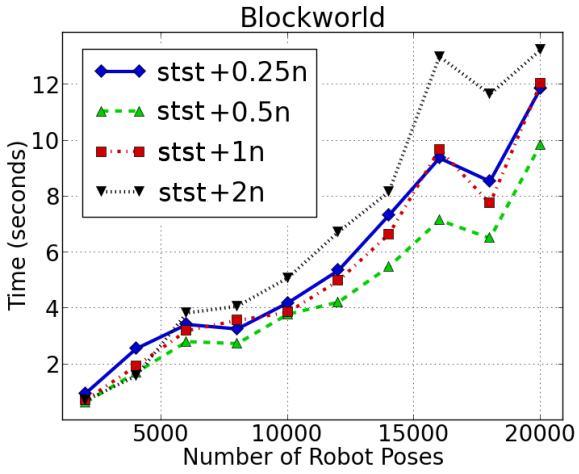


Figure 26: The timing results of stst+cn .

decrease consistently, and (2) stst+cn delivers two to four times better subgraph preconditioners than sptree+cn. These results suggest that our algorithm can produce better subgraph preconditioners.

5.3.5 Timing Results on Synthetic Datasets

I evaluated the running time of using different subgraph preconditioners in the PCGLS method [12] to solve the Blockworld problem and compared the performance against the state-of-the-art sparse direct solver (CHOLMOD [23]).

I generated Blockworld datasets up to twenty thousand robot poses, and ran the

Gauss-Newton algorithm to solve the nonlinear SLAM problem for ten iterations. In each iteration, I used either PCGLS or CHOLMOD to solve the linear systems. For the PCGLS method, I used either sptree+cn or stst+cn as the preconditioners. The stopping criteria for PCGLS are (1) the norm of the current gradient is smaller than 10^{-2} times of the initial gradient, or (2) the number of PCGLS iterations exceeds one thousand. For CHOLMOD, I use the implementation in SuiteSparse compiled with GotoBlas2. All of the solvers run with single thread. Since different settings achieve different errors in the end, I reported the time to achieve the error that is ϵ-close to the optimum, i.e.,

$$\frac{|e - e_*|}{|e_0 - e_*|} \leq \epsilon \tag{46}$$

where e_0 is the initial error, e is the current error, e_* is the minimum achieved error of all solvers, and ϵ is a threshold. I set $\epsilon = 10^{-8}$ in our experiments. Notice that since our algorithm to find a good subgraph is still inefficient, I *excluded* the time spent in Algorithm 1, and focused on comparing the efficiency of linear solvers.

I first evaluated the performance of PCGLS solvers with the stst+cn preconditioners, and showed the results in Figure 26. we can see that the stst+0.5n setting is more efficient than the others. These results suggest that the most efficient subgraph preconditioner is not necessarily the one with the most edges, and finding the right amount of additional edges to augment a spanning tree involves a trade-off: Inserting too few edges into the subgraph may not lead to an effective preconditioner, while inserting too many edges into the subgraph may slow down the overall performance.

Then I compared the PCGLS solvers with CHOLMOD and showed the results in Figure 27. We can see that CHOLMOD is two to three times slower than the PCGLS solvers, and it suggests that direct solvers are not suitable for solving large-scale SLAM problems. Comparing the subgraph preconditioners, we can see that stst+1n is two times faster than sptree+1n. These results suggest that our subgraph preconditioners are more effective.

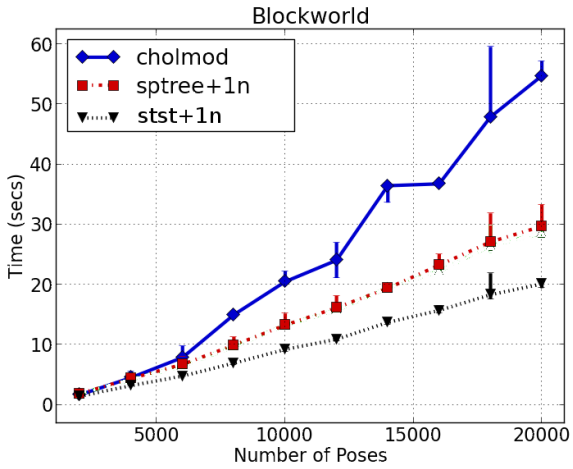


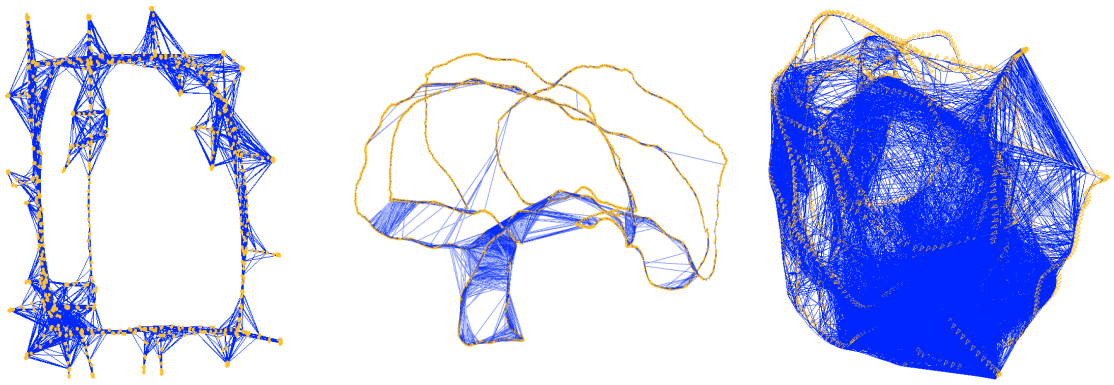
Figure 27: The timing results of different linear solvers.

5.3.6 Timing Results on Real Dataset

I also evaluated the performance of these solvers on two real datasets. Existing public SLAM datasets are mostly sparse graphs which cannot demonstrate the advantages of iterative solvers. To this end, I collected two real datasets of 2,000 images with a Videre STOC camera in an office environment, where the camera constantly visits the same place to create many loop-closure constraints. Figure 28 shows the bird's-eye view of the camera trajectories and the pose constraints of the datasets.

I used the visual odometry pipeline presented in [10] to initialize the robot poses by composing the relative pose constraints along the image sequence, and then used the vocabulary tree technique [83] to generate 33,234 loop-closure constraints.

I used the same algorithm described in the last experiment to solve this dataset and reported the running time in Table 4. I used $c = 1$ as it gave the best performance in this experiment. we can see that although Algorithm 1 is slow, our subgraph preconditioner (stst+1n) can significantly improve the efficiency of solving the problems. More specifically, our solver is 16% and 40% faster than sptree+1n and CHOLMOD respectively in terms of solving the Lab02 problem, and 23% and 57% faster than sptree+1n and CHOLMOD respectively in terms of solving the Cubicle02 problem.



(a) (b) (c)

Figure 28: The bird's-eye view of the (a) Intel, (b) Lab02, and (c) Cubicle02 dataset. The yellow points denote the poses while the blue lines denote the constraints.

Table 1: The timing result on the real datasets in seconds.

Datasets	Poses	Measurements	sptree+1n	stst+1n		CHOLMOD
				Alg. 1	others	
Lab02	1,998	15,505	3.2	3.1	2.7	4.5
Cubicle02	1,998	33,234	4.4	5.9	3.4	7.9

5.4 Improving the Efficiency of Finding STSPs

I have shown the potential of using STSPs to improve the efficiency of solving large SLAM problems, but the limitation is that the MCMC-based algorithm is still too slow in practice. To resolve this problem, I aim to improve the efficiency of finding good subgraph preconditioners by using heuristics and domain knowledge in SLAM. The key observation I make is that in order to find good subgraph preconditioners, it is not imperative to start with the best tree. I use this observation to derive efficient algorithms to construct subgraph preconditioners with quality comparable to STSPs.

5.4.1 Spanning Trees from Heuristics and Domain Knowledge

Since the MCMC-based algorithm to finding a good spanning tree is the bottleneck of finding an STSP, our first step is to replace it with some heuristic spanning tree and examine their performance. Here I consider a few heuristic spanning trees: (1) The

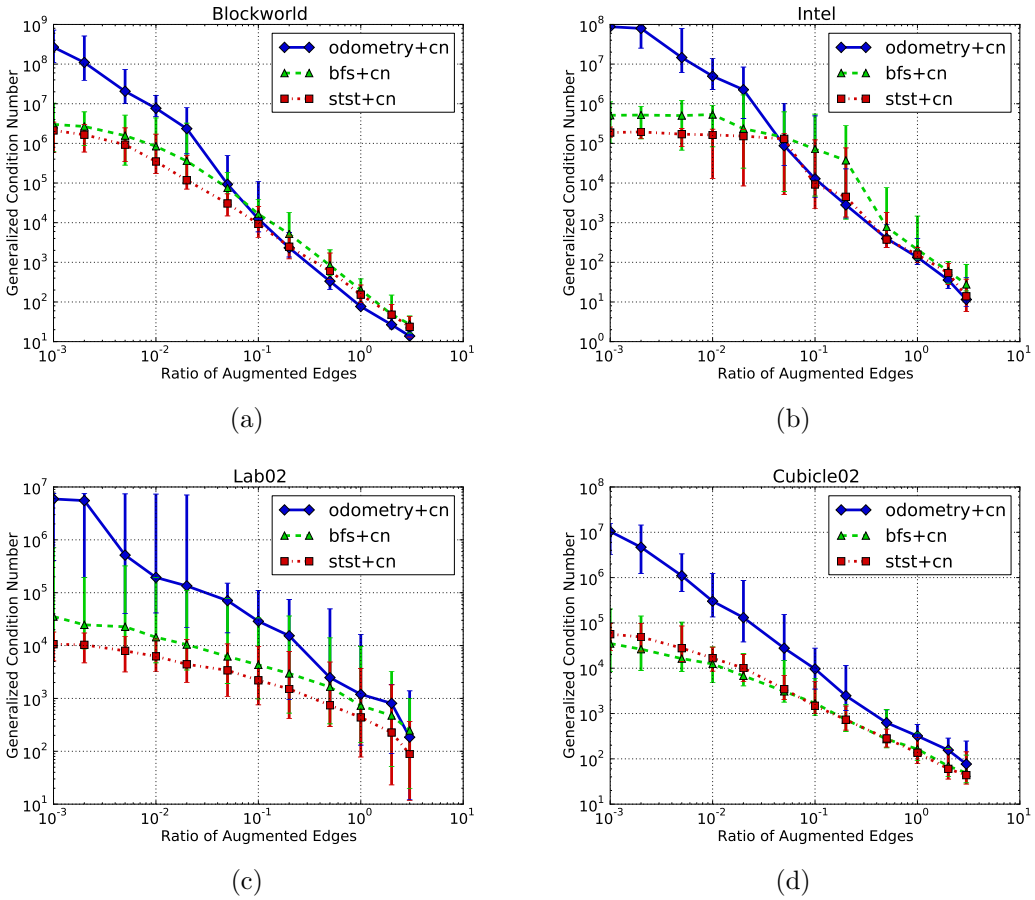


Figure 29: The generalized condition numbers of heuristic subgraphs on (a) Blockworld (b) Intel (c) Lab02 and (d) Cubicle02 datasets.

odometry chains along the robot's trajectory, and (2) The breadth-first-search (BFS) trees (starting a breadth-first search from a random vertex of the graph). The intuition of using these trees is that the odometry chain captures the movement of robots and can serve a natural skeleton of the graph, and a BFS tree has been shown to be a good low-stretch spanning tree for the Laplacian problems [35]. Moreover, these trees can be efficiently computed in $\mathrm{O}(m)$ time where m is the number of edges in the graph. I repeat the same experiments in Section 5.3.4 on the Blockworld, Lab02 and Cubicle02 datasets, and show the results in Figure 29.

We can see that the odometry chain (odometry+cn) starts as a very bad spanning tree at the beginning, but finishes as good subgraph preconditioners for the Blockworld

and Intel datasets. However, its performance is unsatisfactory for the Lab02 and Cubicle02 datasets. I conjecture the main reason is that the first two datasets have more regular graph structures and contain only local loop-closures. These properties make the odometry chain a good skeleton to augment additional edges.

The BFS-based subgraph preconditioners (bfs+cn) have better performance than odometry+cn in the Lab02 and Cubicle02 datasets, but they have consistently worse performance than the support-theoretic subgraph preconditioners (stst+cn). Nevertheless, the gap is typically less than a factor of two for the four datasets. This behavior makes bfs+cn also a good approximation to the stst+cn .

5.5 Summary

In this chapter, I proposed a new metric based on support theory to evaluate the quality of a spanning tree preconditioner for SLAM. I used this metric to develop an MCMC-based algorithm to find good subgraph preconditioners. To the best of our knowledge, this is the first attempt to derive theoretically good subgraph preconditioners for SLAM. Although our MCMC-based algorithm is too slow for practice, I applied heuristics and domain knowledge in SLAM to improve the efficiency of finding good subgraph preconditioners and discussed their tradeoffs. Finally, I applied these subgraph preconditioners to solve synthetic and real SLAM problems, and demonstrated that the proposed subgraph preconditioners display significant improved efficiency.

CHAPTER VI

GENERALIZED SUBGRAPH PRECONDITIONERS

Using preconditioned conjugate gradient (PCG) method to solve large SfM problems has received significant attention recently [105, 4, 5, 19, 51]. However, when applying subgraph preconditioners (SP) to solve these problems, I notice that Hessian-based preconditioning techniques such as the block-Jacobi preconditioner actually have better performance. This phenomenon motivates us to generalize the definition of subgraph preconditioners.

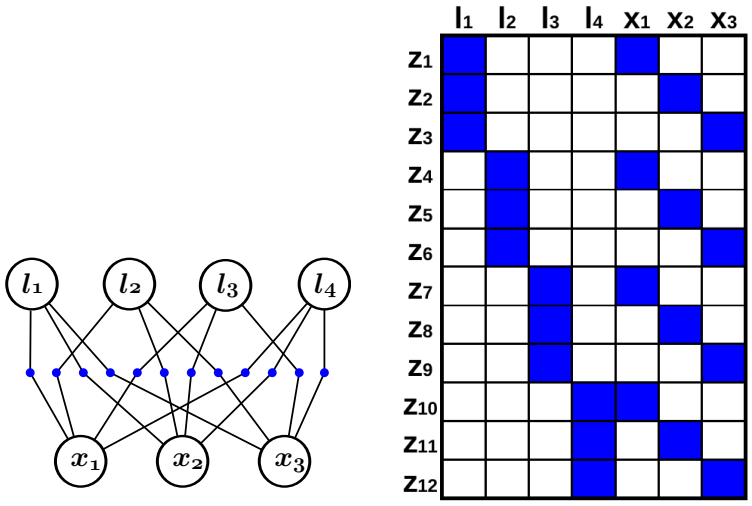
To this end, I propose the Generalized Subgraph Preconditioners (GSP) that adapt subgraph preconditioners for large-scale SfM. While SP picks a subgraph of the *Jacobian* factor graph (Figure 30), GSP operates on the *Hessian* factor graph (Figure 32) which is more general and leads to more effective preconditioners.

In the following, I will explain the Hessian factor graph representation, and how to use it to design good preconditioners for SfM problems. The preliminary results have been published in [53, 54].

6.1 Hessian Factor Graph

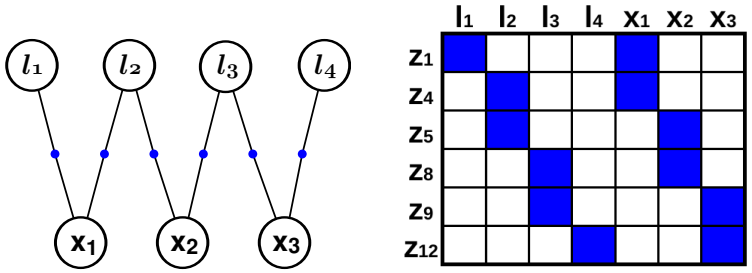
To gain insight into the performance properties of both Jacobi and SP preconditioners, I investigate the structure of the *Hessian* matrix $\mathbf{H} \approx \mathbf{A}^T\mathbf{A}$ appearing in the normal equation (10). The Hessian matrix can also be represented as a graph, more specifically a Gaussian Markov Random Field (GMRF). Every principal sub-matrix of $\mathbf{H}$ corresponds to the information matrix of the conditional distribution given the other variables [29, 78]. In this sense, solving a GMRF is analogous to solving Eq. (10).

Yet a GMRF is usually represented as an undirected graph which is not expressive



(a) (b)

Figure 30: A toy SfM problem with three cameras and four points. (a) The Jacobian factor graph. The vertices denote the camera and the point variables. The blue dots denote the projection factors. (b) The symbolic representation of the Jacobian matrix **A**. Each row denotes a Jacobian factor, and each column indicates a variable.



(a) (b)

Figure 31: An example of a subgraph preconditioner. (a) The Jacobian factor graph that corresponds to a subset of the measurements (sub-problem) in Figure 30. (b) The symbolic matrix representation of the subgraph.

enough for designing subgraph preconditioners. It prompts us to resort to a finer-grained Hessian factor graph representation. The difference is that I create two unary and one binary factors out of each measurement, and accumulate all of them in a *Hessian* factor graph. The number of unary factors attached to a variable equals to the number of the associated measurements, with one binary factor per measurement.

As an example, consider the measurement between x_0 and l_0 in Figure 30(a) and assume A_{x_0} and A_{l_0} are the corresponding block entries in the first row of the

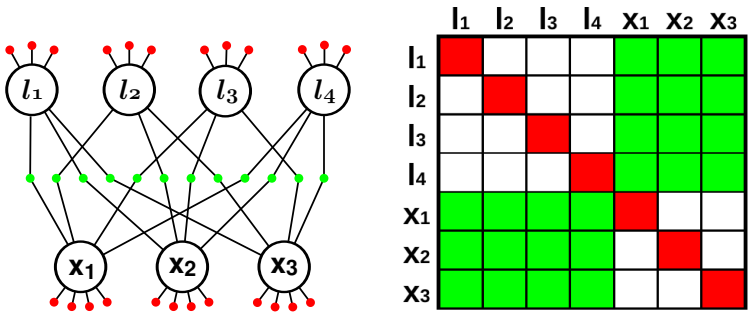


Figure 32: (a) The Hessian factor graph representation of the toy problem in Figure 30, where the red dots denote unary factors while the green dots denote binary factors. This representation resembles to the Gaussian Markov Random Field representation [29, 78]. (b) The symbolic representation of the Hessian matrix $\mathbf{H} \approx \mathbf{A}^T\mathbf{A}$. Both rows and columns indicate variables. A diagonal (red) block indicates the certainty of a variable given the other variables are known. An off-diagonal block indicates whether two variables are correlated given that the other variables are known. Each non-zero off-diagonal (green) block corresponds to a Jacobian factor in Figure 30(a) or a binary Hessian factor in (a).

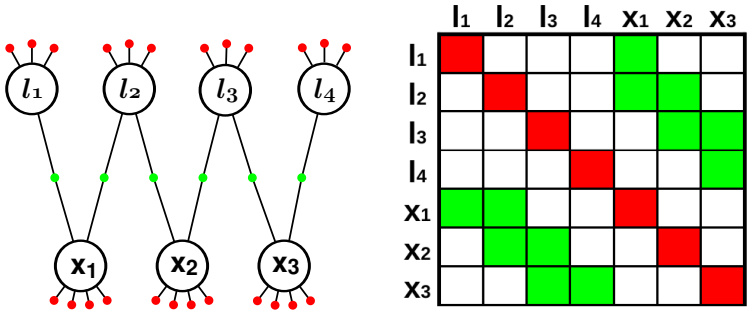


Figure 33: An example preconditioner that GSP can generate but SP cannot.

Jacobian matrix in Figure 30(b). Since the Hessian matrix is the sum of outer product of the block rows of the Jacobian matrix, we can see that this measurement actually corresponds to three terms in the Hessian matrix: $A_{x_0}^T A_{x_0}$, $A_{l_0}^T A_{l_0}$ and $A_{x_0}^T A_{l_0}$. Notice that the first two are unary factors of x_0 and l_0, and the third is a binary factor between them. They encode the information contributed by this measurement to the conditional Gaussian densities. Repeating this process for all measurements, we can build the Hessian factor graph representation illustrated in Figure 32(a).

From this perspective, the problem of designing a good subgraph preconditioner

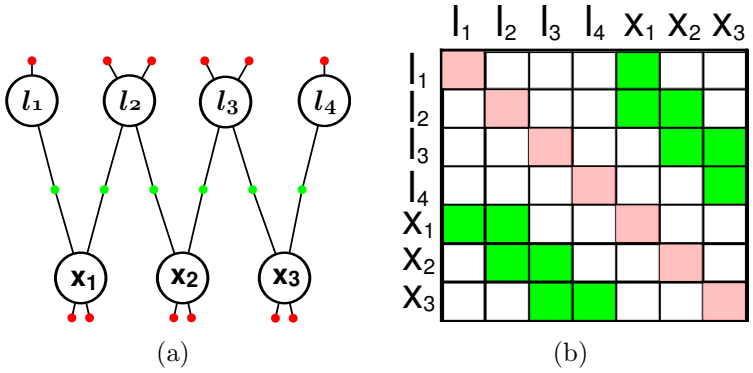


Figure 34: The Hessian factor graph representation of the sub-problem in Figure 31. (a) The Hessian factor graph. (b) The symbolic matrix of the sub-problem. The non-zero off-diagonal blocks are identical to those in Figure 32(b), but the diagonal entries are smaller than those in Figure 32(b). It leads to over-estimating the uncertainty of the variables, especially for the camera variables.

is reduced to picking a subset of Hessian factors from the graph that (1) can be solved efficiently by direct methods, and also (2) make the linear systems well-conditioned. The detail of how to to pick a Hessian factor subgraph will be discussed in the section.

GSP is more expressive than SP because we can always build a Hessian factor graph from a subset of measurements, but not vice versa. For instance, suppose we want to construct a Hessian factor subgraph as in Figure 33 by picking a subset of measurements. One can see that no subset of Jacobian factors in Figure 30(a) corresponds to this Hessian factor graph. Hence the GSP is more general than SP.

The difference between GSP and SP is important for SfM, whose graph is typically an unbalanced bipartite graph. The amount of information that SP brings in for each variable corresponds to the associated measurements in the subgraph. In SfM, if SP picks a spanning tree as the subgraph, then it can only collect at most two out of potentially thousands of unary factors for the camera vertices. This results in over-estimating the uncertainty of the variables and hence leads to unsatisfactory preconditioners. This idea is illustrated in Figure 34. Adding more measurements to the subgraph might help, but it also makes it harder for direct methods to to solve the subgraphs. In contrast, GSP provides the flexibility to keep part or all of the

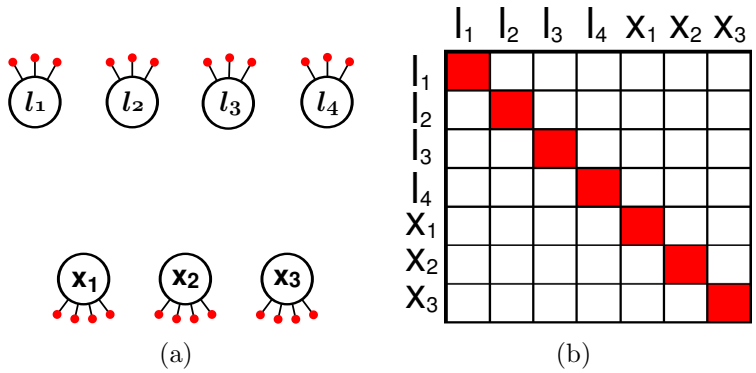


Figure 35: The Jacobi preconditioner of the toy problem.

unary factors (information) for each variable, and hence overcomes this problem.

Notably, the Jacobi preconditioner can be considered as a special Hessian factor subgraph. In the GSP machinery, the Jacobi preconditioner corresponds to picking all of the unary factors and discarding all of the binary factors of in the Hessian factor graph. The idea is illustrated in Figure 35. Note that hereafter when I refer to the Jacobi preconditioner, I actually mean the block Jacobi preconditioner.

6.2 The GSP-n Preconditioners

I propose a greedy algorithm to construct a family of subgraphs with adjustable complexity. On top of these subgraphs, I use GSP to build subgraph preconditioners. The resulting preconditioners are called the GSP-n preconditioners, where n is a parameter that controls the complexity of the subgraph. The SfM graph is a bipartite graph $G = (X, L, E)$, where X denote the camera and L denote the 3-D points vertices on the two sides of G. Each edge in E denotes a measurement that connects the corresponding camera and point vertices.

The goal is to find a subset E_S of E, such that (1) the resulting subgraph G_S has low stretch with respect to G, and (2) the maximum size of the induced cliques does not exceed the predefined parameter n. The maximum size of the induced cliques means the clique number in the factorization phase, which can indirectly affect the computational complexity. A straightforward strategy would be to use a low-stretch

spanning tree of G as the subgraph, but this strategy is sub-optimal because it does not exploit the bipartite and unbalanced nature of G.

Here I introduce some notations to facilitate the exposition. I denote $X(l)$ as the set of cameras associated with a 3-D point l, and $E(l)$ as the corresponding set of edges (measurements). Note that by picking t edges from $E(l)$ into the subgraph, I will induce a clique of size t between the corresponding cameras after eliminating the 3-D point l in the factorization phase. If the edges and the elimination ordering are not chosen appropriately, even larger cliques will appear in the factorization phase.

Here I describe a greedy algorithm to construct a family of subgraphs. First, I build a camera graph G_X where the vertices consist of all cameras and the edge weight between two cameras is defined as the number of 3-D points that are observed by both of them. Then I find a low-stretch spanning tree T_X in G_X [6]. The tree T_X aims to preserve the structural information of G, and provides a skeleton to augment additional edges.

Second, I augment additional edges to the subgraph. Suppose initially the edge set E_S is empty. For each point l, I sort $X(l)$ according to their average distance to the other cameras in $X(l)$ with respect to T_X. Then I pick the edges of $E(l)$ into the subgraph according to this ordering. An edge is added into E_S if it does not induce a camera clique of size greater than n. To this end, I also maintain an array (initially set to 0, whose length is the number of cameras) which holds the size of the maximum clique that a camera belongs to. The array is updated whenever an edge is added. Repeating this process for all 3-D points results in edge set E_S.

Finally I construct the GSP-n preconditioner by using all of the unary factors in the original graph and the binary factors corresponding to the edge set E_S. Note that there are two interesting special cases of the GSP-n preconditioners: GSP-0 corresponds to the Jacobi preconditioner while GSP-∞ corresponds to using the original graph to construct the subgraph preconditioner.

6.2.1 The Symmetry and Positive Definiteness

Being symmetric and positive definite (spd) is a necessary condition for being a valid preconditioner in the conjugate gradient method. Here I show that any GSP-n preconditioner is spd. First, we know that any $\mathbf{H} \approx \mathbf{A}^T\mathbf{A}$ matrix is always spd, and hence GSP-n is also symmetric by construction. Second, discarding off-diagonal block pairs in the Hessian while leaving the block-diagonal unchanged is equivalent to replacing a binary factor by two unary factors in the Jacobian factor graph. The replaced binary factor corresponds to $\mathbf{A}$'s block-row with nonzero blocks A_{x_0} and A_{l_0}, while each new unary factor contains exactly one of these blocks. The inner product of the new factor matrix with itself is spd, which guarantees the validity of GSP preconditioners. Note that discarding symmetrical off-diagonal entries of an *arbitrary* spd matrix may not produce a spd matrix. In the scalar case, Boman et al. [14] proved that matrices with this property must admit a factorization $\mathbf{A}^T\mathbf{A}$, with $\mathbf{A}$ having a factor width ≤ 2.

6.2.2 Results

Here I compared the sparse factorization method (DBA) and the PCGLS method (Appendix A) with three preconditioners: (1) the block Jacobi preconditioner (Jacobi), (2) the subgraph preconditioner (SP), and (3) the generalized subgraph preconditioner (GSP-n). where n denotes the maximum clique size allowed in the greedy algorithm.

I used the Levenberg-Marquardt method as the nonlinear solver. The stopping criteria are (1) the number of iterations exceeds 20, (2) the average reprojection error is less than 0.8 pixel, or (3) the relative decrease of the error is less than 10^{-2}.

For the linear solvers, DBA used CHOLMOD [23] with the COLAMD ordering. For the solvers using the PCGLS method, The stopping criteria are (1) the number of iterations exceeds 2000, (2) the relative decrease of residual is less than 10^{-2}.

For Jacobi, I accumulated all unary factors for each variable (i.e., the diagonal blocks of $\mathbf{A}^T\mathbf{A}$) and solve them independently. For SP, I used the Sparse QR factorization package [27]. For GSP-n , I used the CHOLMOD package [23] with an ordering in which the 3-D points are eliminated first and the cameras are eliminated according to the topological ordering of the camera low-stretch spanning tree. I used Alon et al.'s algorithm to find a low-stretch spanning tree in the camera graph [6]. Note that for SP and GSP-n , the topology of the subgraph is determined at the beginning, and never changed during the optimization. Although GSP offers the flexibility to use various number of unary factors for each variable, I chose to use all of the unary factors for simplicity. In this case, GSP-0 and is mathematically equivalent to Jacobi. Also, GSP-∞ is like DBA if CG runs only one iteration.

I ran the experiments on the BAL datasets released by Agarwal et al. [5]. Since BAL contains many datasets and some of them cannot fit into the memory of a regular PC, I selected ten proper datasets from BAL which have 100K to 500K points (see Table. 3). I run all of the experiments on a Core2 Duo PC with 8G RAM.

6.2.2.1 The Performance of GSP-n

I first investigated the performance of GSP-n for different values of n, and show the timing results in Figure 36. Notice that GSP-n is equivalent to Jacobi when $n = 0$. I excluded the linearization time and focus on comparing the linear solvers. The results show that GSP-n converges faster than Jacobi by 10-30% in most cases.

I also observed that as n increases, the overall time decreases at first, but increases if n is set too high. To better understand the behavior of GSP-n , I broke down the timing results of one dataset and show the major components in Table 2. I can see that as n increases, the subgraph becomes denser and harder to solve, but the time spent on building the subgraph preconditioner is not significant when n is small. Here the important parts are (1) the time to apply the preconditioner per CG

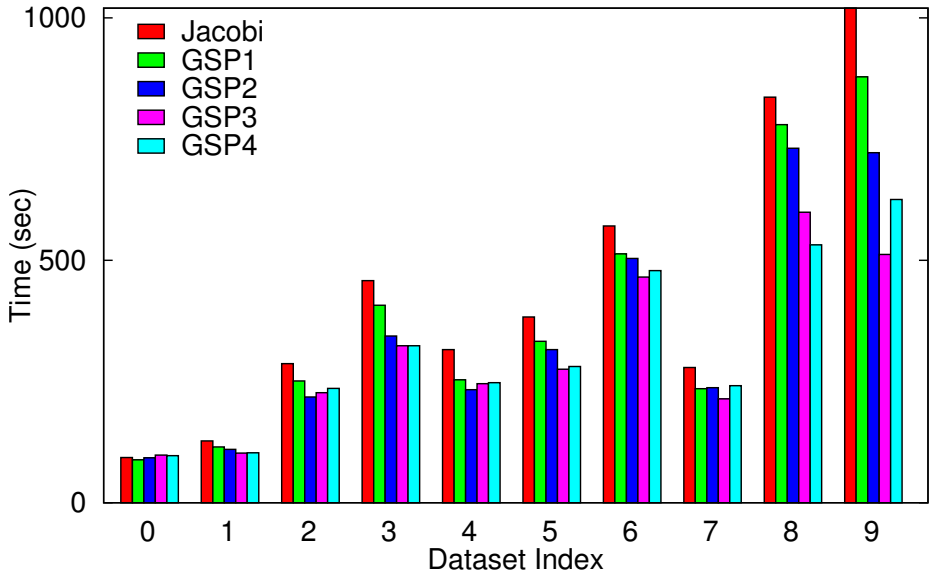


Figure 36: Timing results of Jacobi and GSP-n on BAL.

Table 2: Timing results of GSP-n on the "F-05" dataset. I only show the components relevant to the linear solvers. The columns indicate (1) the maximum clique size in GSP-n , (2) the percentage of edges used in the subgraph, (3) the time of building the subgraph, (4) the time per CG iteration, and (5) the number of total CG iterations, and (6) the total time.

n	edges (%)	build (s)	time/iter (s)	#iters	total (s)
0	0.0	27.2	0.48	1438	732.6
1	19.8	33.4	0.53	1130	648.8
2	26.6	48.7	0.56	866	550.5
3	32.5	69.1	0.62	631	**473.7**
4	39.0	101.5	0.78	526	512.8

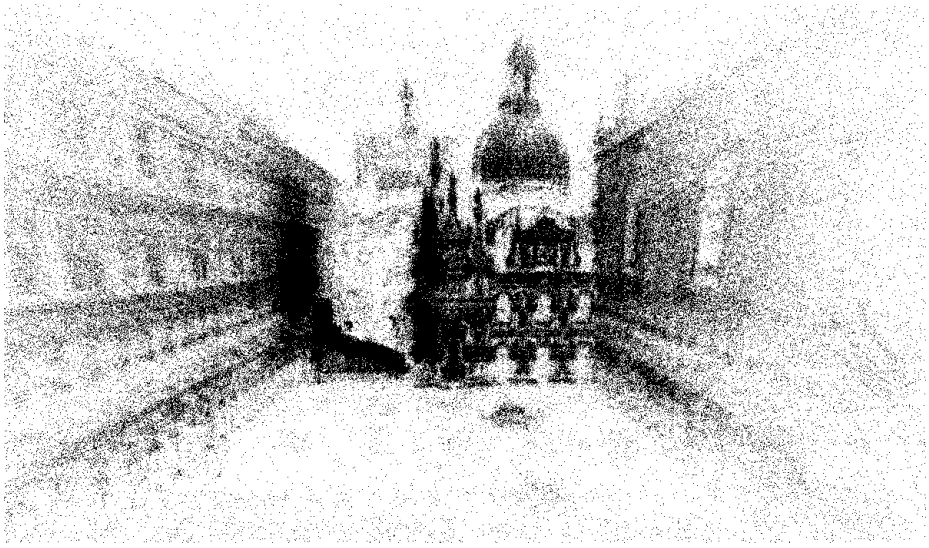
iteration, and (2) the number of total CG iterations. The former increases because the preconditioner becomes denser and hence more computation is involved in the back substitution. The latter decreases because the linear systems become better conditioned. I can see that their product dominate timing and clearly there is a trade-off between these two factors.

Table 3: Timing results (secs) of the four methods on ten BAL datasets. The second column corresponds to the name and index in the original BAL: "D" for "Dubrovnik", "L" for "Ladybug", "V" for "Venice" and "F" for "Final".

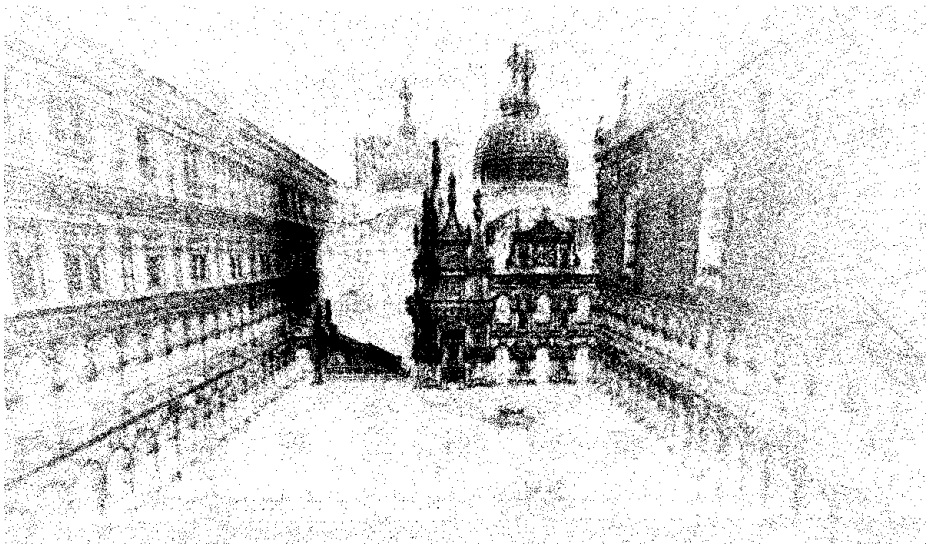
Set	Source	Cameras	Points	Measurements	DBA	Jacobi	SP	GSP-3
0	V-01	89	110,973	562,976	**42**	84	401	89
1	F-01	394	100,368	534,408	**79**	113	256	96
2	V-02	245	198,739	1,091,386	**155**	245	415	196
3	D-15	356	226,730	1,255,268	**187**	397	804	285
4	V-03	427	310,384	1,699,145	313	273	695	**212**
5	L-30	1,723	156,502	678,718	578	312	718	**223**
6	V-04	744	543,562	3,058,863	886	506	913	**407**
7	F-03	961	187,103	1,692,975	1148	252	741	**191**
8	F-02	871	527,480	2,785,977	1939	776	1154	**564**
9	F-05	3,068	310,854	1,653,812	3504	894	2035	**473**

6.2.2.2 Timing Results

Here I compared the timing results of four linear solvers on the BAL datasets. I used $n = 3$ to build subgraphs for both SP and GSP-n . The timing results in Table 3 are sorted according to the DBA time, which reflects the intrinsic difficulty of the datasets. The results confirm that sparse direct methods are efficient for small datasets, but iterative methods are better alternatives for large datasets. Comparing Jacobi and GSP , the results show that by adding extra factors to the subgraph, GSP provide better preconditioners than Jacobi in most of the cases. Comparing SP and GSP , the results show that being able to add more unary factors to the graph is crucial to improve the convergence speed of the conjugate gradient method. An example of the result is shown in Figure 37.



(a)



(b)

Figure 37: Visualization of the "F-03" datasets. The solutions obtained from solving (a) the subgraph and (b) the original graph. The solution to subgraph serves as a good preconditioner to solve the original problem.

6.3 Generalized Subgraph Preconditioners for Reduced Camera Systems

In this section, I aim to adapt our results developed so far to improve the efficiency of solving large reduced camera systems (RCS) [105]. This is in contrast to the results presented in Section 6.2, which designs novel subgraph preconditioners for solving the

original factor graph (i.e. the normal equation). To this end, I first investigate the state-of-the-art preconditioner for solving the RCSs and then improve its quality by using the generalized subgraph preconditioners, support-theoretic techniques and a novel reweighting scheme.

6.3.1 Reduced Camera Systems

SfM graphs are typically unbalance bipartite graphs where one part consists of the cameras and the other consists of the points. The size of the point part is much larger than the camera part. In this case, working on the reduced camera system is beneficial because eliminating all the points can be done efficiently and the number of variables in the new system is also significantly reduced.

Here I derive the reduced camera system. Assume we have the following linear least-squares problem obtained from linearizing the bundle adjustment problem:

$$\begin{bmatrix} \mathbf{A}_p & \mathbf{A}_c \end{bmatrix} \begin{bmatrix} \mathbf{x}_p \\ \mathbf{x}_c \end{bmatrix} = \mathbf{b} \tag{47}$$

where $\mathbf{A}_p$ and $\mathbf{A}_c$ denote the Jacobians associated with the points and cameras respectively, and $\mathbf{x}_p$ and $\mathbf{x}_c$ correspond to the point and camera variables. By multiplying the transpose of the Jacobian matrix on both sides, we can obtained

$$\begin{bmatrix} \mathbf{B} & \mathbf{E} \\ \mathbf{E}^T & \mathbf{C} \end{bmatrix} \begin{bmatrix} \mathbf{x}_p \\ \mathbf{x}_c \end{bmatrix} = \begin{bmatrix} \mathbf{d}_p \\ \mathbf{d}_c \end{bmatrix} \tag{48}$$

where $\mathbf{B} = \mathbf{A}_p^T \mathbf{A}_p$, $\mathbf{C} = \mathbf{A}_c^T \mathbf{A}_c$, $\mathbf{E} = \mathbf{A}_p^T \mathbf{A}_c$, $\mathbf{d}_p = \mathbf{A}_p^T \mathbf{b}$ and $\mathbf{d}_c = \mathbf{A}_c^T \mathbf{b}$. If the graph is bipartite, we can see that $\mathbf{B}$ and $\mathbf{C}$ are block diagonal matrices, and $\mathbf{E}$ encodes the problem structure. To derive the reduced camera system, we can eliminate the points by left-multiplying a matrix

$$\begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{E}^T \mathbf{B}^{-1} & \mathbf{I} \end{bmatrix} \tag{49}$$

on both sides to obtain

$$\begin{bmatrix} \mathbf{B} & \mathbf{E} \\ \mathbf{0} & \mathbf{C} - \mathbf{E}^T\mathbf{B}^{-1}\mathbf{E} \end{bmatrix} \begin{bmatrix} \mathbf{x}_p \\ \mathbf{x}_c \end{bmatrix} = \begin{bmatrix} \mathbf{d}_p \\ \mathbf{d}_c - \mathbf{E}^T\mathbf{B}^{-1}\mathbf{d}_p \end{bmatrix} \tag{50}$$

The part associated with the camera variables

$$(\mathbf{C} - \mathbf{E}^T\mathbf{B}^{-1}\mathbf{E})\mathbf{x}_c = \mathbf{d}_c - \mathbf{E}^T\mathbf{B}^{-1}\mathbf{d}_p \tag{51}$$

is called the reduced camera system which is typically solved by using preconditioned conjugate gradient method for large SfM problems. Once the optimal $\mathbf{x}_c^*$ is obtained, the optimal $\mathbf{x}_p^*$ can be computed in linear time by

$$\mathbf{x}_p^* = \mathbf{B}^{-1}(\mathbf{d}_p - \mathbf{E}\mathbf{x}_c^*), \tag{52}$$

where $\mathbf{B}^{-1}$ can be applied efficienctly because $\mathbf{B}$ is a block diagonal matrix.

Note that for large problems, the reduced camera system is rarely explicitly built because it might take too much resource to compute and store the matrix [5]. Moreover, the matrix-vector multiplication for the system matrix $\mathbf{C} - \mathbf{E}^T\mathbf{B}^{-1}\mathbf{E}$ can be implemented as four matrix-vector multiplications and one vector-vector subtraction. This technique makes it possible for the conjugate gradient method to solve the reduced camera system without explicitly computing the matrix. Yet it also makes the preconditioners based on incomplete factorization unsuitable for this scenario.

6.3.2 Visibility-Based Preconditioners

Kushal and Agarwal [70] proposed to use visibility-based preconditioners (VBP). The main idea is to use the number of points shared by two cameras to define the edge weights in the camera graph, and use these weights to identify camera clusters. More specifically, for the ith camera we can define a binary visibility vector $v_i \in \mathbb{R}^n$ where n is the number of points in the problem. The entry $v_i(j)$ equals to 1 if the jth point is observed by the ith camera, otherwise $v_i(j)$ equals to 0. Then we can use the

visibility vectors to define the similarity score between two cameras as

$$w(i,j) = \frac{v_i \cdot v_j}{|v_i| \cdot |v_j|}. \tag{53}$$

We can further use (53) to define a graph structure on the reduced camera system, where a vertex denotes a camera, and an edge between two vertices indicates that two cameras observe common 3D points, and the corresponding edge weight is set to their similarity score defined in (53).

Kushal and Agarwal [70] used the canonical view algorithm [91] to cluster the cameras, and proposed two variations of the visibility-based preconditioners based on the camera clusters. The first uses only the edges within the camera clusters to build the cluster-Jacobi preconditioner. The second used the maximum-weighted degree-2 forest of the clustered graph to build a cluster-tridiagonal preconditioner. Once the edges of the cluster graph are identified, they computed the corresponding entries in the reduced camera system and factorized it to build the preconditioner.

6.3.3 Generalized Subgraph Preconditioners for Reduced Camera Systems

I aim to build generalized subgraph preconditioners based on the ideas in VBPs. Instead of using fixed structures of the camera cluster graph, my main idea is to relax this limitation and use a *subgraph* of the camera cluster graph as the preconditioner. In this sense, the visibility-based preconditioners by Kushal and Agarwal can be considered as special cases of our new preconditioners.

Note that the techniques developed in Chapter 5 cannot be directly applied to this setting because of two reasons. First, the reduced camera system is in Hessian form, and therefore does not have the incidence graph structure in the Jacobian matrix. Second, our support-theoretic techniques assume the entries in matrix are available, but the reduced camera system for large problems are not explicitly computed.

One way to to overcome this problem is to identify a subgraph based on a scalar-weighted clustered camera graph and use it to build a generalized subgraph preconditioner. For each edge in the clustered graph graph, I assign a scalar weight to it based on the number of common landmarks between two camera clusters, without using the actual values of the reduced camera matrix. Then I use a low-stretch spanning subgraph of the scalar-weighted camera graph to build generalized subgraph preconditioners. Similarly, once the required entries in the reduced camera system are identified, they will be computed to build the preconditioner. The effectiveness of this new preconditioner will be demonstrated in Section 6.3.5.

6.3.4 The Positive Definiteness

A valid preconditioner for the conjugate gradient method has to be positive definite, but taking a subgraph of the reduced camera graph does not guarantee that the resulting preconditioner is always positive definite. This is different from the cases I investigated in Chapters 4, 6 and 5 where the positive definiteness is guaranteed. One way to enforce the positive definiteness is via the property of strictly diagonally dominant.

Definition 5 (Strictly Diagonally Dominant)**.** *A matrix is strictly diagonally dominant if for every row and column of the matrix, the magnitude of the diagonal entry is larger than or equal to the sum of the off-diagonal entries of that row or column. More specifically, a matrix* $\mathbf{M} \in \mathbb{R}^{n \times n}$ *is strictly diagonally dominant if*

$$\|\mathbf{M}_{ii}\| > \sum_{j \neq i} |\mathbf{M}_{ij}|, \quad \forall \quad i \tag{54}$$

Notably, it can be proved that a symmetric and strictly diagonally dominant matrix is always positive definite via the Gershgorin circle theorem [41]. To enforce the positive definiteness of the cluster-tridiagonal preconditioners, Kushal and Agarwal [70] used a fixed ratio, 0.5, to down-scale the off-diagonal entries of the original

matrix when a non-positive pivot is encountered during the sparse Cholesky factorization. They used this ratio because it guarantees the positive definiteness of the new preconditioner matrix. Yet it might significantly degrade the quality of preconditioners because this strategy essentially ignore half of the conditional correlation between the variables.

In contrast, I propose to use a conservative strategy for creating better generalized subgraph preconditioners. Here I summarize the key steps. Similarly, a sparse Cholesky method is used to factorize the preconditioner at the beginning. Whenever a negative pivot is encountered during the elimination process, I simply down-scale the off-diagonal entries by a more conservative ratio, e.g. 0.95, and re-apply the sparse Cholesky factorization to the reweighted matrix. Rescaling and refactorizing the preconditioner can be done efficiently because the subgraph matrix is typically sparse and the elimination tree for sparse Cholesky factorization can been computed beforehand. Note that there exists alternative strategies to enforce the positive definiteness [88], and they can be adapted to improve the current scheme.

6.3.5 Results

I compared the performance of different preconditioners on solving the reduced camera systems of large SfM problems from the Venice datasets [5]. The preconditioners in comparison are (1) the block-Jacobi preconditioner (Block-Jacobi), (2) the cluster-Jacobi preconditioner (Cluster-Jacobi) [70], (3) the cluster-tridiagonal preconditioner (Cluster-Tridiagonal) [70], (4) the cluster-tree preconditioner (Cluster-Tree), and (5) the cluster-subgraph preconditioner (Cluster-Subgraph). The Block-Jacobi preconditioner used the block diagonal of the reduced camera system. For the other preconditioners, I first ran the canonical view algorithm [91] to cluster the cameras, and then used different subgraph structures of the clustered camera graph. The Cluster-Jacobi

augments Block-Jacobi with the edges within the clusters. The Cluster-Tridiagonal augments Cluster-Jacobi with the edges of the maximum weighted degree-2 forest of the clustered graph. The Cluster-Tree takes a low-stretch spanning tree of the clustered graph. The Cluster-Subgraph augments Cluster-Tree with extra high-stretch edges.

I used the following procedures to solve the problems. First, I used the initial estimates from the BAL datasets and ran thirty iterations of the Levenberg-Marquardt

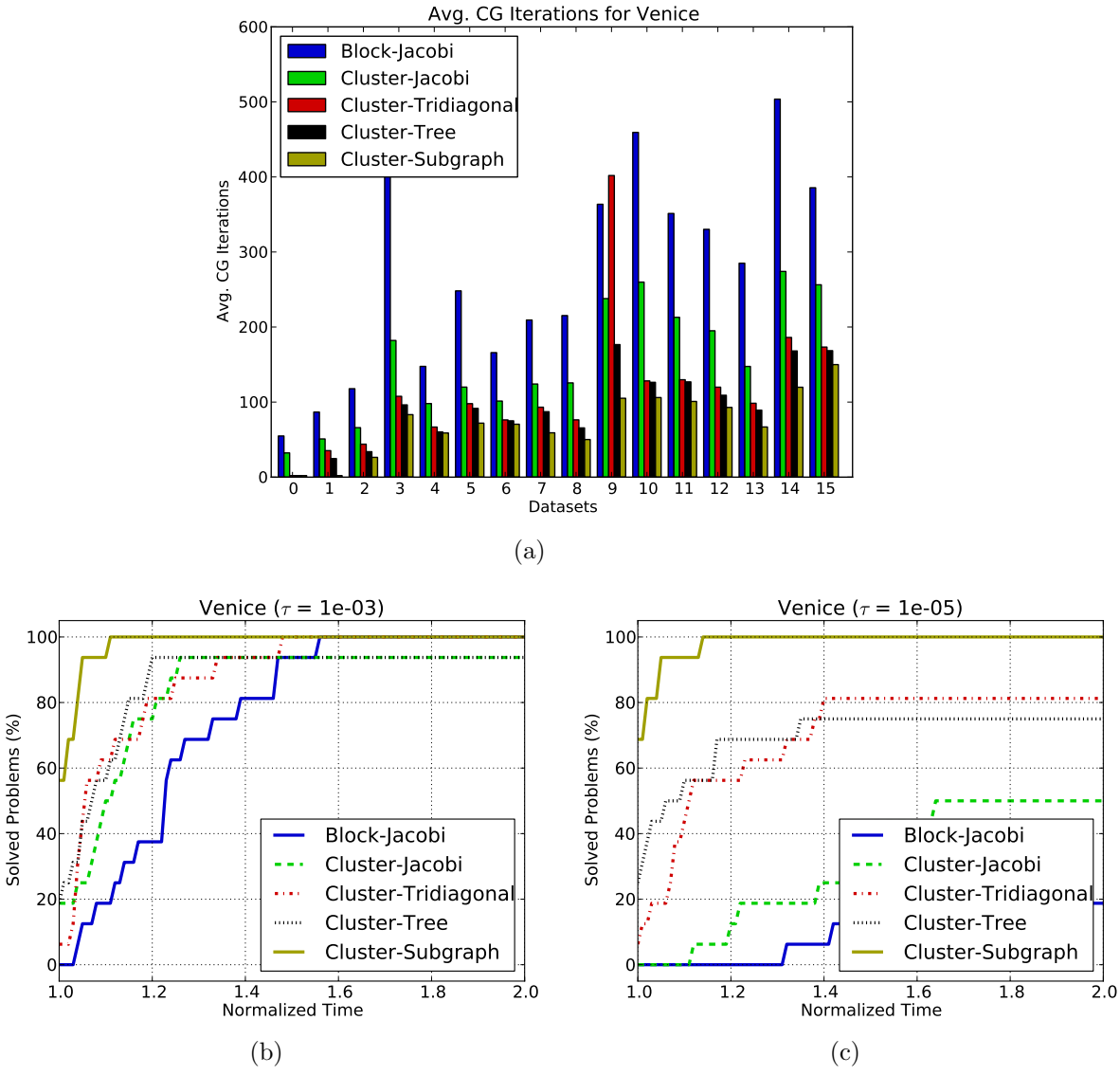


Figure 38: The results on the Venice datasets with a strict threshold for PCG. (a) The total number of itertions of the conjugate gradient method. (b) (c) The performance profile of solving the problems with $\tau = 10^{-3}$ and $\tau = 10^{-5}$.

algorithm to reduce the error. In each iteration, I used the preconditioned conjugate gradient method to solve the reduced camera system with the above preconditioners. I used two stopping thresholds for PCG to compare the preconditioners.

In the first experiment, I used a strict threshold ($\epsilon = 10^{-4}$) for PCG to compare the quality of different preconditioners in terms of solving the *linearized* bundle adjustment problems. I first showed the number of average iteration counts in Figure 38(a).

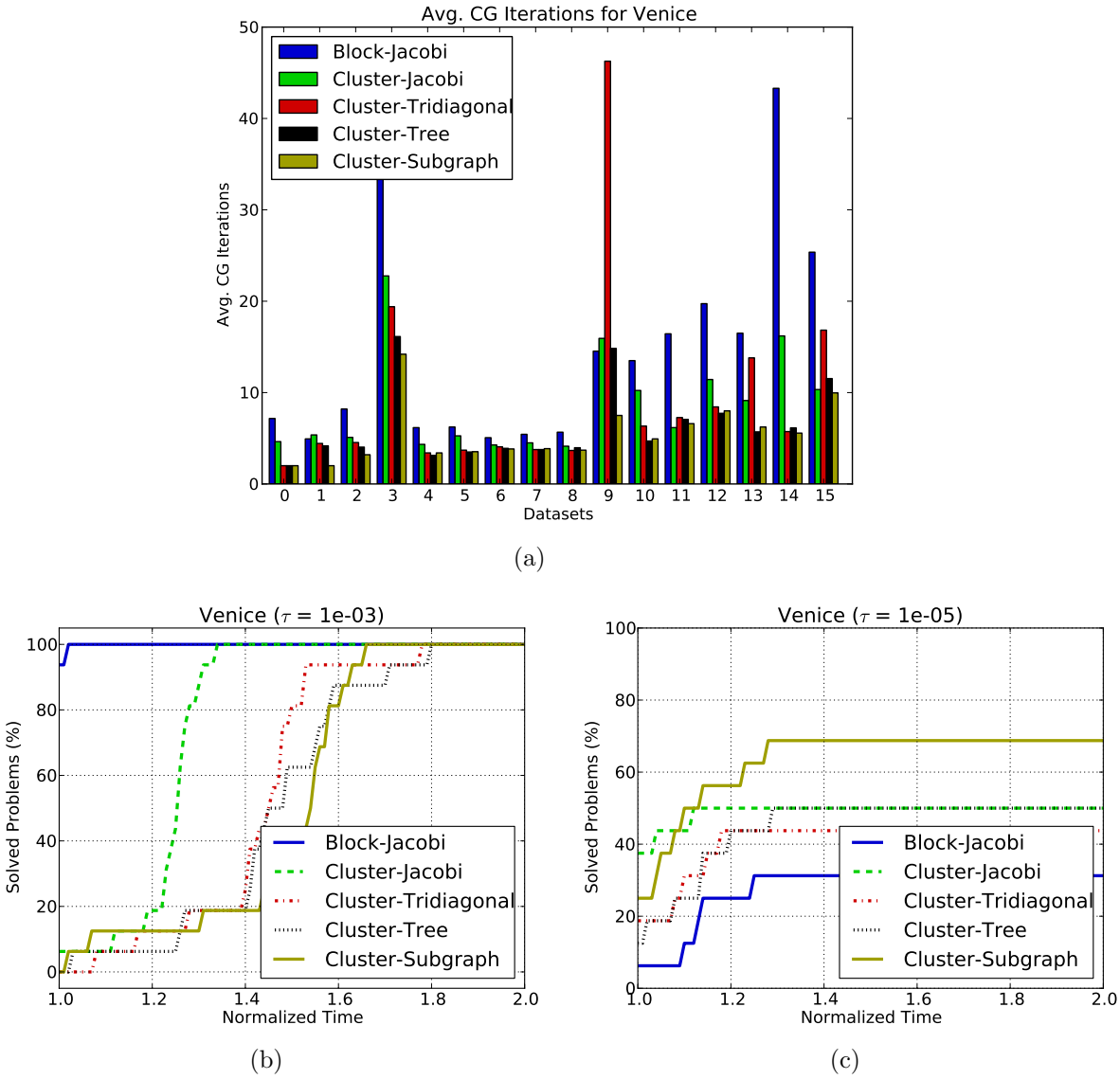


Figure 39: The results on the Venice datasets with a loose threshold for PCG. (a) The total number of iterations of the conjugate gradient method. (b) (c) The performance profile of solving the problems with $\tau = 10^{-3}$ and $\tau = 10^{-5}$.

We can see that the Block-Jacobi and Cluster-Jacobi preconditioners generally have worse performance because they require more iterations to converge in most of the datasets. The Cluster-Tridiagonal is better than the first two preconditioners in most of the datasets, but its performance may degrade significantly for some datasets, e.g. the 9th dataset. After looking closely at the results, I noticed that it happens when it encounters a non-positive definite matrix and requires to down-scale the off-diagonal entries. The Cluster-Tree and Cluster-Subgraph have better performance because they require consistently fewer iterations than the others. The results confirm that adding more edges improve the quality of the preconditioners. In addition, our conservative reweighting schemes not only enforce the positive definiteness, but also prevent the quality of subgraph preconditioners from degrading too rapidly.

I also compared the performance profile [31] of different preconditioners in terms of solving the whole problems and showed the results in Figures 38(b) and 38(c). Note that I excluded the time of clustering the cameras because its performance can be improved independently. We can see that the results generally match what we observed in terms of the iteration counts. The Cluster-Subgraph preconditioner shows as a clear winner over the others.

In the second experiment, I repeated the previous experiment with a loose threshold ($(\epsilon = 10^{-1})$) for PCG and showed the results in Figure 39. We can see that in Figure 39(a) that the number of required average iterations are much fewer than the previous experiment. I observed that in most of the cases, PCG can converge in less than fifteen iterations. It means that the time spent in PCG is much less than that in the previous experiment.

I also showed the performance profile in Figures 39(b) and 39(c). We can see that the simple Block-Jacobi preconditioner is a clear winner in Figure 39(b) because for the other complicated preconditioners, the cost to build the preconditioners exceeds the gain in the iteration counts of PCG. Nevertheless, we can also see from Figure 39(c)

that using subgraph preconditioners can lead to smaller error in the end.

6.4 Summary

While direct methods are efficient for small datasets and iterative methods are more appropriate if the memory requirement is of concern, a subgraph-based preconditioning method combines their advantages and provides a better alternative for solving large-scale SfM. One such method is SPCG, which to the best of our knowledge has not been applied to the SfM problem. Although for large datasets SPCG is significantly better than direct methods and the plain CG method, its behavior is sub-optimal: as the SfM graph is bipartite and unbalanced, SPCG over-estimates the uncertainty of the variables. In contrast, GSP avoids this problem, and is more expressive and suitable for SfM. Well-known preconditioners like Jacobi fit naturally in the GSP context. To exploit the graphical structure of the problem, I develop an efficient algorithm to construct a family of generalized subgraph preconditioners. When applied to large datasets, the GSP-n preconditioners display promising performance.

Moreoever I propose to use generalized subgraph preconditioners on the clustered camera graphs to improve the efficiency of solving large-scale SfM problems. Compared to previous work using a similar strategy, my key observation is to use general subgraphs instead of fixed structures. I also propose a conservative reweighting scheme to avoid degrading the quality of preconditioners too rapidly. I evaluated the performance of our new preconditioners on the BAL datasets with two thresholds for PCG. I observed that our subgraph preconditioners are more effective when a tight threshold is used or when a high-quality solution is needed. Yet when a loose threshold is used, the cost of building the subgraph preconditioners exceeds the gain in the iteration counts of PCG. These observations suggest that we can have a better preconditioner-selection strategy according to the thresholds used in PCG.

CHAPTER VII

INCREMENTAL SPCG (ISPCG)

So far I have demonsrated the effectiveness of subgraph preconditioners to solve large-scale *batch* SLAM and SfM problems. In this chapter I aim to extend the applicability of this machinery to solve large-scale *online* SLAM problems. To this end, I propose the incremental subgraph-preconditioned conjugate gradient method as the first step toward this goal. This new method is built based on two state-of-the-art SLAM methods, incremental smoothing and mapping (iSAM) [58] and SPCG [30]. I show that the proposed method is consistent, efficient and can find the optimal solution. Promising results were obtained on large-scale simulated and real SLAM problems.

7.1 Overview

Modern SLAM methods often formulate SLAM as a graph-based optimization problem [77, 29, 69]. The state-of-the-art *online* SLAM methods stem from incremental smoothing and mapping (iSAM) [58], and hierarchical optimization (HOG-Man) [44]. Yet these methods do not scale well when there are many loop-closures because they all aim to incrementally or hierarchically factorize the information matrix which is expensive for large and complex problems. In SLAM, a loop-closure refers to an event that the robot recognizes a previously mapped area. Loop-closures are essential to limit the growth of uncertainty and improve the estimate. Here we are interested in solving online SLAM problems with many loop-closures, which are common when the robot has long-range sensors, e.g., cameras, and explores in large open space.

Many techniques have been proposed to efficiently solve SLAM problems with many loop-closures. Methods based on graph sparsification aim to reduce the problem size by discarding redundant edges, marginalizing redundant vertices, or using

convex optimization techniques, but they lead to either inconsistent estimate or high computation complexity [64, 34, 50, 56, 49, 66, 20]. Iterative methods have been used to solve the SLAM problems with many loop-closures, but they either assume all measurements are available in advance or require good initialization to obtain good results [32, 48, 39, 85, 43, 30, 52].

In this chapter, I propose a new method, incremental subgraph-preconditioned conjugate (iSPCG) method, to efficiently solve online SLAM with many loop-closures. The main idea is to use iSAM to incrementally solve a sparse subgraph to obtain an approximate solution. When the error grows larger than a threshold or the optimal solution is requested, I apply subgraph-preconditioned conjugate gradient method (SPCG) [30, 52] to solve the entire graph to obtain the optimal solution. Note that the subgraph preconditioner and the initial estimate for SPCG are provided by iSAM. Then I use the optimal solution from SPCG to regularize iSAM's estimates in the following steps. Unlike previous work leading to information loss or inconsistent estimate [34, 50, 66, 20], I show that iSPCG is consistent, efficient and optimal.

This method has the following contributions: (1) I propose iSPCG to efficiently solve online SLAM problems with many loop-closures by combining the advantages of two state-of-the-art SLAM methods. (2) I provide a theoretical analysis of iSPCG and show that it can provide consistent and optimal estimate. (3) I apply iSPCG to solve large-scale simulated and real SLAM problems and obtain promising results.

7.2 Incremental SPCG (iSPCG)

Here I present the incremental subgraph-preconditioned conjugate gradient (iSPCG) method that combines two state-of-the-art techniques, iSAM and SPCG, to efficiently solve online SLAM problems with many loop-closures. iSAM is one of the state-of-the-art method for online SLAM problem. The main idea is to perform fast incremental updates of the square root information matrix when new measurements are

added [59]. Recently, it has been shown that Bayes trees provide a better machinery and allow incremental reordering and just-in-time relinearization [58]. Generally speaking, iSAM can achieves constant computational complexity when the robot is exploring the environment. These desirable properties make iSAM one of the state-of-the-art methods for online SLAM problems. Yet the performance degrades when there are many loop-closures because iSAM will have to frequently reorder the Bayes Tree as well as update the corresponding conditional densities.

To address this problem, I use iSAM to incrementally solve a *sparse* subgraph to obtain an approximate solution. When the error on the remaining part grows larger than a threshold or the optimal solution is requested, I apply SPCG [30, 52] to solve the entire graph to obtain the optimal solution. Note that the subgraph preconditioner and the initial estimate for SPCG are provided by iSAM. Then I use the optimal solution from SPCG to regularize iSAM's estimates in the following steps. The detail of each step will be explained in the following sections.

7.2.1 Solving Subgraphs with iSAM

Consider a SLAM problem as a factor graph $G = (V, E)$, where V denotes the robot poses, and E denotes the measurements (factors). The graph is incrementally separated into two parts: the *subgraph* part $H = (V, E_H)$ and the *constraint* part $C = (V, E_C)$. Then iSAM is used to incrementally solve the subgraph part, and constantly keep an approximate solution. To evaluate the quality of the approximate solution, I compute the normalized chi-square error on the subgraph part χ^2_H, the constraint part χ^2_C and the entire graph χ^2_G, respectively. If the error is small, i.e.,

$$\chi^2_G \leq \tau_g \quad \text{and} \quad \frac{\chi^2_C}{\chi^2_H} \leq \tau_r, \tag{55}$$

where τ_g and τ_r are thresholds, I accept iSAM's solution and go to the next iteration. Note that this scheme is efficient if the subgraph is sparse because the associated Bayes Tree and conditional densities will only be updated occasionally.

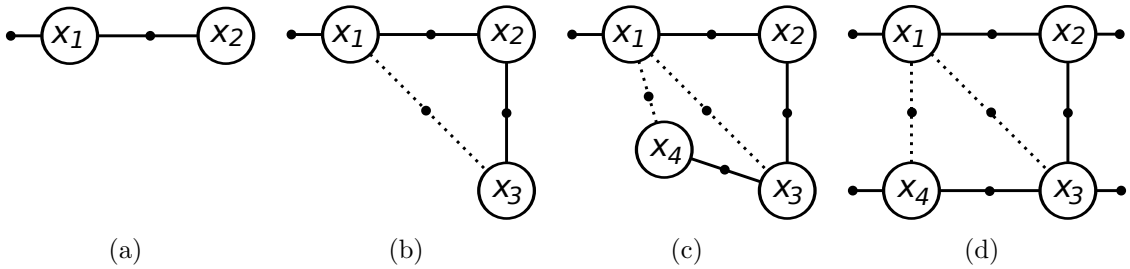


Figure 40: Illustration of the iSPCG method on a simple graph. The solid factors belong to the subgraph while the dashed factors correspond to the remaining part. (a) Initially the graph is still sparse. Hence all factors belong to the subgraph, and iSAM can solve it very efficiently. (b) There is one loop-closure constraint, but leaving it out of the subgraph does not introduce significant error. (c) There are more loop-closures, but this time leaving them out of the subgraph leads to unsatisfactory results. Hence SPCG is invoked to optimize the entire graph. (d). The solution obtained from SPCG is used to regularized iSAM in the next iterations.

7.2.2 Solving Original Graphs with SPCG

If iSAM's approximate solution leads to high error or the optimal solution is requested by the user, I apply SPCG to solve the entire graph with iSAM's solution as the initial estimate and iSAM's factorization of the approximate information matrix as the subgraph preconditioner. Since iSAM's solution is typically close to the true solution, and the subgraph preconditioner can effectively reparametrize the problem, therefore SPCG will converge to the optimal solution in a few iterations.

7.2.3 Regularizing iSAM with SPCG's Solutions

Finally, iSAM has to be informed of the optimal solution from SPCG, otherwise it will drift again in the next iterations. To this end, I add prior factors to the subgraph

$$e_p(\theta_k) = \frac{1}{2}\|\theta_k - \theta_k^{spcg}\|^2_{\Sigma_k} \quad \forall \quad k = 1, 2, ..., n \tag{56}$$

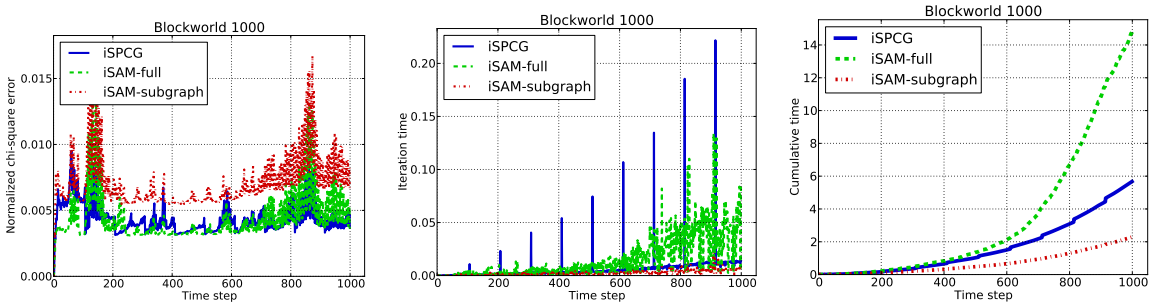
where θ_k^{spcg} denotes SPCG's solution of the kth variable, Σ_k is the covariance matrix of the prior factor, and n is the number of variables. Note that these prior factors only exist in iSAM and will not affect the optimal solution, and they will be replaced after next invocation of SPCG. The key steps of iSPCG are illustrated with a simple

Algorithm 2: One step of the iSPCG algorithm

Input: G_{t-1} is the current factor graph, and H_{t-1} is a subgraph of G_{t-1}. F_t denotes new factors.

Output: new estimate θ_t

1. split $F_t = F_t^H \bigcup F_t^C$ into subgraph and constraints parts
2. use iSAM to solve the new subgraph $H_t = H_{t-1} \bigcup F_t^H$
3. **if** *iSAM's solution* θ_t^{isam} *is acceptable* **then return** θ_t^{isam}
4. use SPCG to solve the new graph $G_t = G_{t-1} \bigcup F_t$ with θ_t^{isam} as initial estimate
5. use SPCG's solution θ_t^{spcg} to regularize iSAM hereafter
6. **return** θ_t^{spcg}



(a) (b) (c)

Figure 41: The results on a synthetic Blockworld dataset with 1,000 poses and 20,000 measurements. (a) The normalized chi-square error. (b) The processing time per time step. (c) The cumulative processing time.

example in Fig. 40 and summarized in Algorithm 2.

7.3 The Consistency of iSPCG

Here we prove a sufficient condition that iSPCG is consistent. The consistency of an online SLAM method is important because it can prevent us from being over-confident about the current estimates and therefore help us make conservative data association in the SLAM frontend. While previous work used convex optimization techniques to enforce the consistency [107, 49], here we show that the proposed method can be proved to be consistent.

We start by defining the notion of consistency:

Definition 6 (Consistency)**.** *The estimate of the mean and covariance* $\mathbf{\Sigma}$ *of Gaussian*

random variables is consistent if

$$E[\hat{\boldsymbol{\mu}} - \boldsymbol{\mu}] = 0 \tag{57}$$

$$\hat{\boldsymbol{\Sigma}} \succeq \boldsymbol{\Sigma} \tag{58}$$

where $(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ *denote the true values, and* $(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}})$ *denote the estimate [9].*

Proving (57) is typically simple due to the Gaussian assumption and the central limit theorem. Therefore, we focus on proving (58). We need the following four lemmas to prove the consistency of iSPCG.

Lemma 7. *If* $\mathbf{X}$ *and* $\mathbf{Y}$ *are symmetric and positive-definite matrices, then* $\mathbf{X} \succeq \mathbf{Y}$ *if and only if* $\rho(\mathbf{X}^{-1}\mathbf{Y}) \leq 1$*, where* $\rho(\cdot)$ *denotes the largest eigenvalue [47, p.471].*

Lemma 8. *If* $\mathbf{X}$ *and* $\mathbf{Y}$ *are positive-definite matrices. If* $\mathbf{X} \succeq \mathbf{Y}$*, then* $\mathbf{X}^{-1} \preceq \mathbf{Y}^{-1}$*.*

Proof. Since $\mathbf{X} \succeq \mathbf{Y}$, according to Lemma 7, we know $\rho(\mathbf{X}^{-1}\mathbf{Y}) \leq 1$. Since $\rho(\mathbf{X}^{-1}\mathbf{Y}) = \rho(\mathbf{Y}\mathbf{X}^{-1})$, therefore we obtain $\rho(\mathbf{Y}\mathbf{X}^{-1}) \leq 1$ and then $\mathbf{X}^{-1} \preceq \mathbf{Y}^{-1}$. □

Lemma 9. *Given a real symmetric matrix* $\mathbf{X} \in \mathbb{R}^{n \times n}$*, and its eigenvalues* $\{\lambda_i\}_{i=1}^n$*, then the eigenvalues of* $(\mathbf{X} + t\mathbf{I})$ *are* $\{\lambda_i + t\}_{i=1}^n$*.*

Proof. Suppose v_i is the ith eigenvector of $\mathbf{X}$, then we obtain $(\mathbf{X}+t\mathbf{I})v_i = \mathbf{X}v_i + t\mathbf{I}v_i = \lambda_i v_i + tv_i = (\lambda_i + t)v_i$ □

Lemma 10. *Solving a subgraph is consistent.*

Proof. Consider rearranging the Jacobian matrix in (9) into

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_H \\ \mathbf{A}_C \end{bmatrix} \tag{59}$$

where $\mathbf{A}_H$ denotes the Jacobians associated to the subgraph, and $\mathbf{A}_C$ denotes that of the remaining part. Suppose $\boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1}$ denotes the information matrix, we need to prove that $\boldsymbol{\Sigma}_H \succeq \boldsymbol{\Sigma}$. From (59), we can see that $\boldsymbol{\Lambda} - \boldsymbol{\Lambda}_H = \mathbf{A}^T\mathbf{A} - \mathbf{A}_H^T\mathbf{A}_H = \mathbf{A}_C^T\mathbf{A}_C \succeq \mathbf{0}$. Therefore, $\boldsymbol{\Lambda} \succeq \boldsymbol{\Lambda}_H$. Using Lemma 8, we can obtain $\boldsymbol{\Sigma}_H = \boldsymbol{\Lambda}_H^{-1} \succeq \boldsymbol{\Lambda}^{-1} = \boldsymbol{\Sigma}$. □

Lemma 11. *Adding the regularization terms in (56) to a subgraph maintains the consistency if* $\mathbf{\Sigma}_k = t^{-1}\mathbf{I}$ *and* $t \leq \lambda_{min}(\mathbf{\Lambda}_C)$*, where* $\lambda_{min}(\cdot)$ *denotes the smallest eigenvalue.*

Proof. The information matrix of the regularized subgraph is $(\mathbf{\Lambda}_H + t\mathbf{I})$. To prove the lemma, we need to show $\mathbf{\Lambda} \succeq (\mathbf{\Lambda}_H + t\mathbf{I})$. We can see that $\mathbf{\Lambda} - (\mathbf{\Lambda}_H + t\mathbf{I}) = (\mathbf{\Lambda}_H + \mathbf{\Lambda}_C) - (\mathbf{\Lambda}_H + t\mathbf{I}) = (\mathbf{\Lambda}_C - t\mathbf{I})$. Suppose the eigenvalues of $\mathbf{\Lambda}_C$ are $\{\lambda_i\}_{i=1}^n$. Using Lemma 9, we know the eigenvalues of $(\mathbf{\Lambda}_C - t\mathbf{I})$ are $\{\lambda_i - t\}_{i=1}^n$. Since $t \leq \min_i \lambda_i$, we know $\lambda_i - t \geq 0$, for all i. Hence $(\mathbf{\Lambda}_C - t\mathbf{I}) \succeq \mathbf{0}$ and $\mathbf{\Lambda} \succeq (\mathbf{\Lambda}_H + t\mathbf{I})$. □

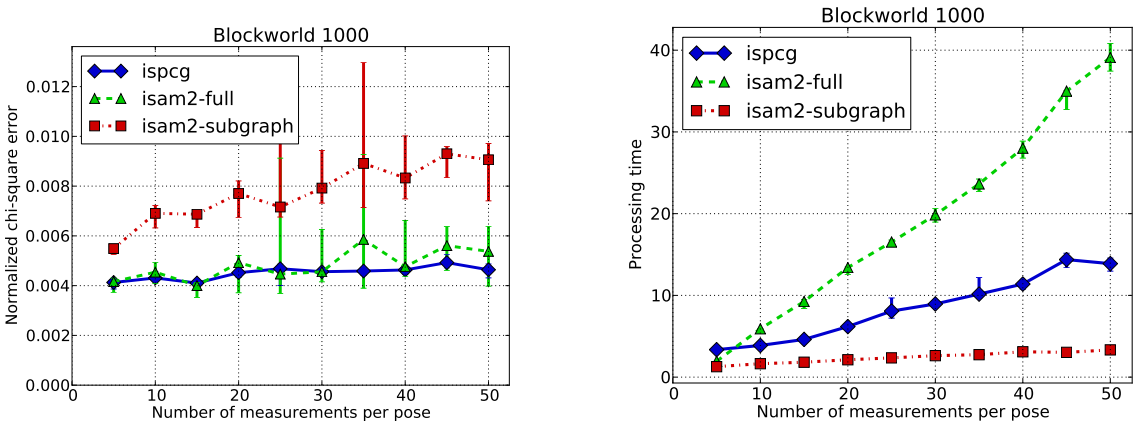
Corollary 12. *iSPCG gives consistent and optimal estimates.*

Proof. The discussion can be splitted into two parts: (1) For the iSAM part, using Lemma 10, we know that using iSAM to solve a subgraph always leads to consistent estimates. Moreover, using iSAM to solve a *regularized* subgraph also leads to consistent estimates if we assign the covariance matrices according to Lemma 11. The solution from iSAM is optimal in the sense that the normalized chi-square error is always smaller than a predefined threshold. (2) For the SPCG part, the estimates are both consistent and optimal because they are obtained by solving the original graph.

Note that in Lemma 11 we assume the linearization points for both the subgraph and the original graph are identical, but this assumption is not always true for nonlinear SLAM problems. Nevertheless, the difference between two linearization points is bounded because iSAM would relinearize whenever there are sufficient changes in the current estimates. □

7.4 Results

I conducted experiments to evaluate the accuracy, speed and scalability of iSPCG, and compare it with iSAM [58] on simulated and real datasets. For iSPCG, I used a subgraph consisting of the odometry chain of the robot poses plus n randomly



(a) (b)

Figure 42: The results on Blockworld datasets with different number of loop-closures.

selected edges, where n is the number of robot poses. Such a simple choice has shown its effectiveness in [52]. The thresholds in (55) are empirically set to $\tau_g = 10^{-2}$ and $\tau_r = 5.0$ respectively. I also used inverse iteration method [104] to estimate the smallest eigenvalue to determine the proper covariance matrices of the regularization terms in (56). For iSAM, I used the implementation in GTSAM [1] with default parameters. I ran all of the experiments with single thread on a PC with an Intel Core i7 CPU.

7.4.1 Simulated Datasets

To facilitate the comparison, I generated a number of synthetic Blockworld problems, simulating a robot traversing a block world. The bird's-eye view of this problem is illustrated in Fig. 12. For each robot pose, I added various number of constraints to its closest neighbors, and these measurements are contaminated by zero-mean and normally distributed noise.To make the SLAM problem well-posed, I attached a prior factor to the first robot pose.

7.4.1.1 Accuracy

I evaluated the accuracy of different solvers on a Blockworld problem with 1,000 poses and 20,000 measurements, and showed the results in Fig. 41. Note that "iSAM-full"

means using iSAM to solve the entire graph, while "iSAM-subgraph" means using iSAM to solve the subgraph as in iSPCG. In Fig. 41(a), we can see that both iSPCG and iSAM-full can achieve lower errors because they both aim to solve the original problem. Yet iSAM-subgraph consistently has larger errors because it only used part of the information. Moreover, in some of our trials, I observed that iSAM-subgraph cannot solve the problems.

7.4.1.2 Timing

I also evaluated the running time of different solvers on the same dataset, and reported the results in Figs. 41(b) and 41(c). I can see iSAM-full quickly becomes expensive because of many loop-closures, which makes it unsuitable for large-scale problems. iSAM-subgraph is very efficient but it also leads to higher errors or potentially wrong solutions. By combining the advantages of iSAM and SPCG, iSPCG can be more than two times faster than iSAM-full and also obtain high-quality solutions.

Notably, from Fig. 41(b), we can see that iSPCG periodically has a spike, which is undesirable for online applications. We observed that this happened when the solution of iSAM is unsatisfactory and SPCG has to be invoked to optimize the full graph. One way to resolve this problem is by splitting iSPCG into two threads: one thread running iSAM in the frontend, and the other thread running SPCG in the backend. In addition, we observed that there is a tradeoff between the quality of solutions obtained from iSPCG as well as the efficiency of iSPCG. That is, the smaller the thresholds τ_g and τ_r, the better solutions we obtain, but the more often we have to run to solve the full graphs. How to automatically determine these thresholds is another interesting question. I plan to explore these two directions in future work.

7.4.1.3 Scalability

I evaluated the performance of different solvers on Blockworld datasets with various number of loop-closures. I reported the tenth percentile, the median and the ninetieth

percentile over twenty trials of the final errors and total processing times in Fig. 42. We can see that iSAM-full consistently achieves lower errors as in the previous experiments because it aims to solve the original problem. iSAM-full is also efficient when the number of loop-closures is small, but quickly becomes expensive when the number of loop-closures increases, which makes it unsuitable for large-scale problems.

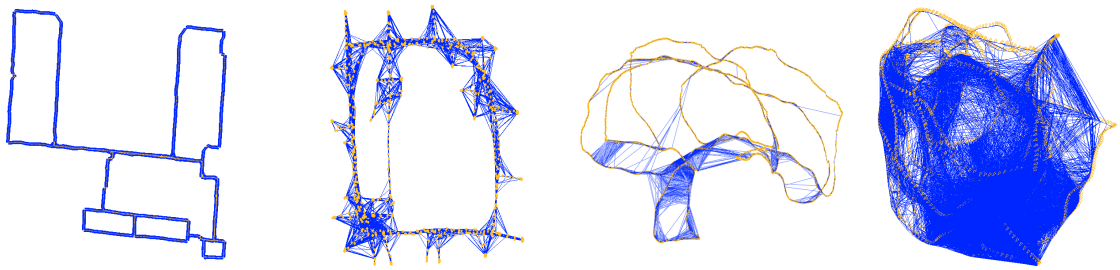
For iSAM-subgraph, since it aims to solve a sparse sub-problem, its efficiency is always good and independent of the number of loop-closures. Yet I observed that its error is not only consistently higher than that of iSAM-full and iSPCG, but also increases with the number of loop-closures. These properties also make iSAM-subgraph unsuitable for obtaining high-quality solutions.

For iSPCG, it can provide high-quality solutions, and also be up to four times faster than iSAM-full. These properties make iSPCG a better choice for large-scale SLAM problems with many loop-closures.

7.4.2 Real Datasets

I also evaluated the performance of iSPCG and iSAM on four real datasets. The "Killian" and "Intel" datasets are publicly available on Radish. The "Lab02" and "Cubicle02" are created by the authors with a Videre STOC camera in an office environment, where the camera constantly visits the same place to create many loop-closure constraints. I used the vocabulary tree technique [83] implemented in [10] to generate loop-closure constraints. The latter two datasets can be downloaded from my website [2].

From the results in Table 4, we can see that iSAM is more efficient when the SLAM problems do not have many loop-closures, i.e., the ratio between the number of measurements to the number of poses is low. I observed that iSAM is up to two times faster in "Killian" and "Intel" datasets. However, when the ratio becomes larger, iSPCG starts to show its advantages. On the "Cubicle02" dataset, I observed



(a) (b) (c) (d)

Figure 43: The real datasets: (a) Killian (b) Intel (c) Lab02 (d) Cubicle02.

Table 4: The timing results on real datasets in seconds. The "Ratio" column indicates the ratio between the number of measurements to the number of poses, which can be an indicator of the difficulty of the problem.

Name	Poses	Measurements	Ratio	iSPCG	iSAM
Killian	1,941	3,995	2.1	4.6	**3.1**
Intel	910	4,454	4.9	2.0	**1.1**
Lab02	1,998	15,505	7.8	**17.5**	18.1
Cubicle02	1,998	33,234	16.6	**69.2**	441.6

that iSPCG is 6.3 times faster than iSAM. I omitted the normalized chi-square errors because both method achieve similar errors for all datasets.

7.5 Related Work

Solutions to the online SLAM problem have been well-studied in literature. Here I focus on recent results of pose graph optimization [77, 29, 69], and refer the readers to [33] and [8] for the developments of filtering-based methods.

One of the main challenges to SLAM methods is scalability. In this chapter, I addressed the scalability to the number of loop-closures. Many techniques have been proposed and they can be divided into the following categories.

The first category aims to build an intermediate representation of the problem so that the estimate can be obtained efficiently, e.g., incremental smoothing and mapping [59, 58], and hierarchical optimization [44]. Although these methods are efficient for sparse problems, they do not scale well when there are many loop-closures.

Table 5: Comparison between different methods for SLAM problems with many loop-closures.

Method	Efficient	Consistent	Optimal
iSAM	×	○	◎
Marginalization	×	○	◎
Vertex removal	○	×	×
Edge removal (subgraph)	○	○	×
Chow-Liu tree approx.	○	×	×
Convex optimization	×	○	×
iSPCG	○	○	○

The main reason is that they all aim to factorize the information matrix which is expensive when the matrix is dense. Nevertheless, the concepts of these techniques are useful, and therefore I design our method based on one of the state-of-the-art methods in this category.

The second category aims to *sparsify* the robot poses. Earlier work selects keyframes or skeleton graphs [60, 64]. Although these techniques can effectively downsize the problem, they typically lead to information loss and inconsistent estimation. Recent work marginalizes redundant robot poses and induces additional constraints (pseudo loop-closures) between the adjacent poses [34, 50, 56, 49, 21]. These techniques can effectively reduce the number of poses and may lead to consistent estimation, but the graphs after marginalization typically become more dense than the original graphs. This implies that marginalization has to stop at some point because it would eventually become expensive due to the increasing size of the associated clique. Therefore one still has to solve a graph with many loop-closures in the end, which is the place the proposed method can be applied.

The third category aims to sparsify the loop-closures. This process can be guided by thresholding the number of loop-closures per robot pose [34], thresholding the expected information gain [50], locally approximating with a Chow-Liu tree in the

information matrix [66, 20]. Yet these techniques lead to either information loss or inconsistent estimate, which is suboptimal. Consistent edge sparsification methods have been proposed, but they require solving a convex optimization problem, which might be too expensive for large-scale problems [49]. In contrast, the proposed method constantly solves a regularized sparse subgraph, which can be done efficiently, and also proven to be consistent.

The fourth category aims to reparametrize the problem so that the solution can be obtained faster. Incremental pose reparametrization over the odometry chain or a spanning tree of the graph has been used to improve the convergence speed of the stochastic gradient descent method [85, 43]. Using sparse subgraphs to precondition the SLAM problems has been shown to be able to effectively improve the convergence speed of the conjugate gradient method [30, 52]. Yet these techniques are designed for batch SLAM problems. Notably, Sibley et al. [90] showed that using relative pose parametrization makes it possible to incremental solve SLAM in constant time.

iSPCG combines the advantages of the first, the third and the fourth categories and is a consistent and efficient method for online SLAM problems with many loop-closures. The solutions are close to optimal in the iSAM steps and optimal in the SPCG steps. The comparison between the above methods is summarized in Table 5.

7.6 Summary

I propose a new method, iSPCG, to efficiently solve online SLAM problems with many loop-closures. iSPCG has the following advantages: (1) iSPCG is efficient because it combines the advantages of two state-of-the-art SLAM methods, iSAM and SPCG. The iSAM part is efficient because it only has to solve sparse subgraphs. The SPCG part is also efficient because it scales well to the number of loop-closures, utilizes the subgraph preconditioners and initial estimate provided by iSAM, and only being invoked whenever necessary. Finally, iSPCG used the optimal solution

from SPCG to regularize iSAM in the next iterations. (2) I prove that iSPCG can be consistent, while in previous work such property is usually not guaranteed or has to be enforced by convex optimization techniques. Although the data association problem is not addressed in this chapter, the consistency of iSPCG actually can help make conservative data association in the SLAM front-end. (3) iSPCG aims to find the optimal solution because it does not discard any measurements. I apply this method to solve large simulated and real SLAM problems with promising results.

There are several directions for future work. The first is to design a new metric to evaluate the quality of a subgraph for iSPCG, and then use this metric to design an algorithm to incrementally find good subgraphs. Intuitively, this metric should consider the computational complexity of using iSAM to solve the subgraph, the quality of the approximate estimate obtained from iSAM, and the quality of subgraph preconditioner for SPCG. The second is to derive more versatile sufficient conditions to guarantee the consistency of iSPCG. The third is to develop an algorithm to automatically decide the thresholds in iSPCG. The algorithm should consider the tradeoff between the quality of the solutions obtained from iSAM and the time spent on running SPCG. At last, we would like to improve the efficiency of iSPCG by utilizing multiple cores on modern CPUs. Similar to the idea in PTAM [60], we can split iSPCG into two threads: one thread running iSAM to obtain the current estimates in the frontend, and the other thread running SPCG to obtain the optimal estimates in the backend.

CHAPTER VIII

DISCUSSIONS

Here I discuss the implementations and several practical issues when solving large-scale SLAM and SfM problems. The contents consist of my experience during the course of working on this dissertation and they are dedicated to the pragmatic readers. Finally I will conclude this dissertation with final thoughts and future work.

8.1 Implementations

The implementation of the support-theoretic subgraph preconditioners consists of three main components: the first is to find good subgraphs, the second is the preconditioned conjugate gradient method as the linear solver, and the third is the nonlinear optimization algorithms such as the Gauss-Newss algorithm or the Levenberg-Marquardt algorithm.

For the first part, I have described the details of the algorithms in the corresponding chapters and they should be straightforward to implement. Regarding to the graph algorithms, existing libraries might be helpful to some extents. Yet since the nature of SLAM and SfM could have arbitrary structure and the edges could have arbitrary arities, I would recommend implementing a graph class that can support hyperedges between the vertices. Then we can have the freedom to derive algorithms to find good subgraph preconditioners for SLAM and SfM.

For the second part, I implemented a templated version of the preconditioned conjugate gradient method and its least-squares variation in C++. This makes it possible to reuse the code in different places. There are two keys to make it efficient. The first key is to consider the sparsity of the graph/matrix and engineer for the peformance of matrix-vector product operation. BLAS libraries can also be helpful

if the users want to start from lower level functions. Another key is to be able to efficiently build the subgraph preconditioners. For this part, I used CHOLMOD, which is the state-of-the-art sparse Cholesky library to factorize the subgraph matrix.

For the third part, I used the GTSAM [1] library, which provides facility based on factor graphs and Bayes networks to solve both batch and online nonlinear optimization problems. For large-scale SLAM problems, I use least-square conjugate gradient method. For large-scale SfM problem, as suggested by Agarwal et al. [5] and Jeong et al. [51], applying the preconditioned conjugate gradient method to solve the implicitly built reduced camera system delivers the best overall performance.

8.2 Practical Issues

I would like to discuss several practical issues in solving large SLAM and SfM problems. The first is how to choose between the sparse direct methods and subgraph-preconditioned conjugate gradient method when the user is given a SLAM or SfM problem. One way to predict the performance of sparse direct methods is via estimating the level of fill-in in the factorized triangular matrix, and it can be done by running a symbolic variable elimination on the original problem. It is possible to exactly estimate the amount of required operation counts and memory space given the results of the symbolic elimination.

Similarly, we can also estimate the performance of the preconditioned conjugate gradient method, whose computation consists of two parts: the first part is building the preconditioner, and the second part is the number of iterations times the computation per iteration. The estimation of the first part can be done in the same way as the sparse direct methods. For the second part, the number of iterations is inversely proportional to the square root of the generalized condition number in the worst case. Since the smallest generalized eigenvalue is always larger than one, this quantity can be efficiently computed by using the power iteration method. The computation cost

per iteration is dominated by the cost of performing matrix-vector multiplications, applying the preconditioners, and computing the inner products. Similarly, the cost can also be estimated via counting the number of nonzero entries in the matrix and the preconditioner and the dimensions of the problems.

The second practical issue is when to use subgraph preconditioners. In my experience, subgraph preconditioners are effective when the condition number of the problem is large, or high-quality solutions are needed. In the other words, if the problem is well-conditioned or a loose threshold is going to be used in the conjugate gradient method, then the cost of building a subgraph preconditioner may exceed the gain it will bring in.

The third practical issue is how to determine the subgraph complexity. With a slight abuse of notation, this problem can be formulated as an optimization problem:

$$\min_{c} \sqrt{\mathsf{GCN}^{-1}(G, H(c))} \cdot (\mathsf{nnz}(G) + 2 \cdot \mathsf{nnz}(R(c)) + \mathsf{factorize}(H(c)) \tag{60}$$

where G denotes the original graph, $H(c)$ denotes a subgraph with its complexity parametrized by c, $R(c)$ denotes the factorized version of $H(c)$, $\mathsf{GCN}(\cdot)$ denotes the generalized condition number, $\mathsf{nnz}(\cdot)$ denotes number of nonzero entries, and $\mathsf{factorize}(\cdot)$ denotes to cost of factorizing the subgraph. Since this cost function is only parametrized by a single variable c, we can evaluate the cost on a few discrete values and choose the c with the smallest cost.

8.3 Conclusions and Future Work

I have demonstrated that support-theoretic subgraph preconditioners and generalized subgraph preconditioners are effective to improve the efficiency of solving large-scale SLAM and SfM problems through theoretical developments and experimental results. In particular, I evaluated the strength of different preconditioning techniques, and showed that subgraph preconditioners are particularly effective for large and ill-conditioned problems or when high-quality solutions are needed. I also presented

novel support-theoretic metrics and algorithms to derive good subgraph preconditioners for several SLAM and SfM problem settings, and they are more effective than state-of-the-art preconditioning techniques. Finally I presented the potential of subgraph preconditioners to be applied to solve large *online* SLAM problems.

There are several directions for future work. The first is to extend the theoretical results to handle hyper factors and under-constrained factors to accommodate more general measurement models and problem settings. The second is to apply the incremental subgraph-preconditioned conjugate gradient method to solve incremental bundle adjustment problems. The third is to apply the proposed techniques to solve the large-scale problems in the other domains. The last interesting direction is to investigate the potential of nonlinear subgraph preconditioning.

APPENDIX A

CONJUGATE GRADIENT METHOD

The conjugate gradient (CG) method is probably the best iterative method to solve *linearized* SLAM and SfM problems (9) because of its efficiency and minimization property. Here I will summarize the CG method with an emphasis on the least-squares problem.

The conjugate gradient (CG) method was developed by Hestenes and Stiefel [46] to solve symmetric and positive definite (spd) systems. The CG method is a special case of the Krylov space methods. Let me make the following definition.

Definition 13. *Suppose* $\mathbf{M} = \mathbf{A}^T\mathbf{A} \in \mathbb{R}^{n\times n}$ *is a matrix and* $\mathbf{c}_0 \in \mathbb{R}^n$ *is a vector. The Krylov space can be defined as*

$$\mathcal{K}_k(\mathbf{M}, \mathbf{c}_0) = span(\mathbf{c}_0, \mathbf{M}\mathbf{c}_0, \cdots, \mathbf{M}^{k-1}\mathbf{c}_0). \tag{61}$$

When applying the CG method to solve a linear least-squares problem

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{62}$$

where $\mathbf{A} \in \mathbb{R}^{m\times n}$ is a rectangular matrix and $\mathbf{b} \in \mathbb{R}^m$ is a vector, the kth iterate $\mathbf{x}^{(k)}$ output by the CG method satisfies the following minimization property. Let $\mathbf{x}^* = \mathbf{A}^\dagger\mathbf{b}$ be the pseudo-inverse solution and $\mathbf{r}^* = \mathbf{b} - \mathbf{A}\mathbf{x}^*$ the corresponding residual. Then $\mathbf{x}^{(k)}$ minimizes the following error function

$$\mathbf{E}(\mathbf{x}^{(k)}) = (\mathbf{x}^* - \mathbf{x}^{(k)})^T(\mathbf{A}^T\mathbf{A})(\mathbf{x}^* - \mathbf{x}^{(k)}) = \|\mathbf{r}^* - \mathbf{r}^{(k)}\|_2^2 \tag{63}$$

over all vectors in the affine subspace

$$\mathbf{x}^{(k)} \in \mathbf{x}^{(0)} + \mathcal{K}_k(\mathbf{A}^T\mathbf{A}, \mathbf{g}^{(0)}), \quad \mathbf{g}^{(0)} = \mathbf{A}^T(\mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}). \tag{64}$$

The above result states that the CG method finds the best estimate in the Krylov space that minimizes the residuals.

The convergence speed of CG can be characterized by the following proposition.

Proposition 14. *Let* $\mathbf{M}$ *be spd and let* $\mathbf{x}^{(k)}$ *be the* k*th CG iterates, then*

$$\frac{\|\mathbf{x}^* - \mathbf{x}^{(k)}\|_{\mathbf{M}}}{\|\mathbf{x}^* - \mathbf{x}^{(0)}\|_{\mathbf{M}}} \leq \max_{z \in \Gamma(\mathbf{M})} |\mathbf{P}_k(z)|. \tag{65}$$

where $\|\mathbf{x}\|_{\mathbf{M}} = \sqrt{\mathbf{x}^T \mathbf{M} \mathbf{x}}$ *denotes the* $\mathbf{M}$*-norm,* $\Gamma(\mathbf{M})$ *denotes the set of eigenvalues of* $\mathbf{M}$*, and* $\mathbf{P}_k$ *be any* k*th degree polynomial such that* $\mathbf{P}_k(0) = 1$*.*

This proposition leads to an upper bound estimate for the number of CG iterations required to reduce the $\mathbf{M}$-norm of the error to a given tolerance. More specifically, this proposition indicates that the convergence speed of CG depends on the best kth order polynomial that passes through all of the eigenvalues of $\mathbf{M}$. It also implies that CG would converge faster if the eigenvalues of $\mathbf{M}$ is clustered because it is likely to find a low-order polynomial to pass through the eigenvalues.

When the entire spectrum of a matrix is hard to predict beforehand, it is possible to have a looser bound on the convergence speed

$$\frac{\|\mathbf{x}^* - \mathbf{x}^{(k)}\|_{\mathbf{M}}}{\|\mathbf{x}^* - \mathbf{x}^{(0)}\|_{\mathbf{M}}} \leq \Big(\frac{\sqrt{\kappa(\mathbf{M})} - 1}{\sqrt{\kappa(\mathbf{M})} + 1} \Big)^k \tag{66}$$

where $\kappa(\mathbf{M}) = \lambda_n / \lambda_1$ is the condition number of $\mathbf{M}$ that is defined as the ratio between the large eigenvalue to the smallest eigenvalue. This inequality indicates that the larger the condition number, the slower the convergence speed. Moreover, there is a simple connection between the convergence speed of the residual and that of the error induced by $\mathbf{M}$-norm:

$$\frac{\|\mathbf{r}^{(k)}\|_2}{\|\mathbf{r}^{(0)}\|_2} = \frac{\|\mathbf{b} - \mathbf{M}\mathbf{x}^{(k)}\|_2}{\|\mathbf{b} - \mathbf{M}\mathbf{x}^{(0)}\|_2} = \frac{\|\mathbf{M}(\mathbf{x}^* - \mathbf{x}^{(k)})\|_2}{\|\mathbf{M}(\mathbf{x}^* - \mathbf{x}^{(0)})\|_2} \leq \sqrt{\frac{\lambda_n}{\lambda_1}} \frac{\|\mathbf{x}^* - \mathbf{x}^{(k)}\|_{\mathbf{M}}}{\|\mathbf{x}^* - \mathbf{x}^{(0)}\|_{\mathbf{M}}}. \tag{67}$$

Finally, we summarize the least-squares variant of the conjugate gradient (LSCG) method in Algorithm 3.

Algorithm 3: Conjugate Gradient Least-Squares Method

Input: let $\mathbf{x}^{(0)}$ be an initial, and ϵ be the tolerance
$\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}$, $\mathbf{p}^{(0)} = \mathbf{g}^{(0)} = \mathbf{A}^T\mathbf{r}^{(0)}$, $\gamma_0 = \|\mathbf{g}^{(0)}\|_2^2$
for $k = 0$ **to** maximum iterations **do**
  if $\gamma_k < \epsilon$ **then** break
  $\mathbf{q}^{(k)} = \mathbf{A}\mathbf{p}^{(k)}$
  $\alpha_k = \gamma_k / \|\mathbf{q}^{(k)}\|_2^2$
  $\mathbf{x}^{(k+1)} = \mathbf{x}_{(k)} + \alpha_k \mathbf{p}^{(k)}$
  $\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha_k \mathbf{q}^{(k)}$
  $\mathbf{g}^{(k+1)} = \mathbf{A}^T\mathbf{r}^{(k+1)}$
  $\gamma_{k+1} = \|\mathbf{g}^{(k+1)}\|_2^2$
  $\beta_k = \gamma_{k+1}/\gamma_k$
  $\mathbf{p}^{(k+1)} = \mathbf{g}^{(k+1)} + \beta_k \mathbf{p}^{(k)}$
end

A.1 Preconditioned Conjugate Gradient Method

The conjugate gradient method is inefficient when the spectrum of $\mathbf{M}$ is wide-spread or the condition number is large. To improve the convergence speed, we can solve another least-squares problem with the same solution, but with better spectrum:

$$\mathbf{A}\mathbf{R}^{-1}\mathbf{y} = \mathbf{b} \tag{68}$$

where $\mathbf{x} = \mathbf{R}^{-1}\mathbf{y}$. Similarly, I summarize the least-squares variant of the preconditioned conjugate gradient (PCGLS) method in Algorithm 4.

In this appendix I presented the key properties of the conjugate gradient method without rigorous derivations. A more detailed presentation can be found in [12, 87].

Algorithm 4: Preconditioned Conjugate Gradient Least-Squares Method

Input: let $\mathbf{x}^{(0)}$ be an initial, and ϵ be the tolerance
$\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}$, $\mathbf{p}^{(0)} = \mathbf{g}^{(0)} = \mathbf{R}^{-T}(\mathbf{A}^T\mathbf{r}^{(0)})$, $\gamma_0 = \|\mathbf{g}^{(0)}\|_2^2$
for $k = 0$ **to** maximum iterations **do**
  if $\gamma_k < \epsilon$ **then** break
  $\mathbf{t}^{(k)} = \mathbf{R}^{-1}\mathbf{p}^{(k)}$
  $\mathbf{q}^{(k)} = \mathbf{A}\mathbf{t}^{(k)}$
  $\alpha_k = \gamma_k / \|\mathbf{q}^{(k)}\|_2^2$
  $\mathbf{x}^{(k+1)} = \mathbf{x}_{(k)} + \alpha_k \mathbf{t}^{(k)}$
  $\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha_k \mathbf{q}^{(k)}$
  $\mathbf{g}^{(k+1)} = \mathbf{R}^{-T}(\mathbf{A}^T\mathbf{r}^{(k+1)})$
  $\gamma_{k+1} = \|\mathbf{g}^{(k+1)}\|_2^2$
  $\beta_k = \gamma_{k+1}/\gamma_k$
  $\mathbf{p}^{(k+1)} = \mathbf{g}^{(k+1)} + \beta_k \mathbf{p}^{(k)}$
end

REFERENCES

[1] "GTSAM `https://collab.cc.gatech.edu/borg/gtsam`."

[2] "`http://www.cc.gatech.edu/~yjian6`."

[3] Abraham, I. and Neiman, O., "Using petal-decompositions to build a low stretch spanning tree," in *ACM Symp. on Theory of Computing (STOC)*, pp. 395–406, ACM, 2012.

[4] Agarwal, S., Snavely, N., Simon, I., Seitz, S., and Szeliski, R., "Building Rome in a Day," in *Proc. IEEE 12th International Conference on Computer Vision*, 2009.

[5] Agarwal, S., Snavely, N., Seitz, S. M., and Szeliski, R., "Bundle Adjustment in the Large," in *Proc. 10th European Conference on Computer Vision*, 2010.

[6] Alon, N., Karp, R., Peleg, D., and West, D., "A Graph-Theoretic Game and Its Application to the k-Server Problem," *SIAM Journal on Computing*, vol. 24, pp. 78–100, February 1995.

[7] Amestoy, P., Davis, T., Duff, I., and others, "An approximate minimum degree ordering algorithm," *SIAM Journal on Matrix Analysis and Applications*, vol. 17, no. 4, pp. 886–905, 1996.

[8] Bailey, T. and Durrant-Whyte, H., "Simultaneous Localisation and Mapping (SLAM): Part II State of the Art," *Robotics & Automation Magazine*, Sep 2006.

[9] Bar-Shalom, Y., Li, X. R., and Kirubarajan, T., *Estimation with applications to tracking and navigation: theory algorithms and software*. John Wiley & Sons, 2001.

[10] Beall, C., Lawrence, B., Ila, V., and Dellaert, F., "3D reconstruction of underwater structures," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2010.

[11] Bern, M., Gilbert, J., Hendrickson, B., Nguyen, N., and Toledo, S., "Support-graph preconditioners," *SIAM Journal on Matrix Analysis and Applications*, vol. 27, no. 4, pp. 930–951, 2006.

[12] Björck, Å., *Numerical Methods for Least Squares Problems*. SIAM, 1996.

[13] Boman, E. and Hendrickson, B., "On spanning tree preconditioners," *Manuscript, Sandia National Laboratories*, 2001.

[14] Boman, E., Chen, D., Parekh, O., and Toledo, S., "On factor width and symmetric H-matrices," *Linear algebra and its applications*, vol. 405, pp. 239–248, 2005.

[15] Boman, E. and Hendrickson, B., "Support theory for preconditioning," *SIAM Journal on Matrix Analysis and Applications*, vol. 25, no. 3, pp. 694–717, 2003.

[16] Briggs, W., McCormick, S., and others, *A multigrid tutorial*, vol. 72. Society for Industrial Applied Mathematics, 2000.

[17] Brown, D. C., *A solution to the general problem of multiple station analytical stereotriangulation*. D. Brown Associates, Incorporated, 1958.

[18] Byröd, M. and Åström, K., "Bundle Adjustment using Conjugate Gradients with Multiscale Preconditioning," in *Proc. 20th British Machine Vision Conference*, 2009.

[19] Byröd, M. and Åström, K., "Conjugate Gradient Bundle Adjustment," in *Proc. 11th European Conference on Computer Vision*, 2010.

[20] Carlevaris-Bianco, N. and Eustice, R. M., "Generic factor-based node marginalization and edge sparsification for pose-graph slam," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, May 2013.

[21] Carlevaris-Bianco, N. and Eustice, R. M., "Long-term simultaneous localization and mapping with generic linear constraint node removal," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, November 2013.

[22] Chen, D. and Toledo, S., "Vaidya's preconditioners: Implementation and experimental study," *Electronic Transactions on Numerical Analysis*, vol. 16, pp. 30–49, 2003.

[23] Chen, Y., Davis, T., Hager, W., and Rajamanickam, S., "Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate," *ACM Transactions on Mathematical Software*, vol. 35, no. 3, pp. 1–14, 2009.

[24] Chung, F., *Spectral graph theory*, vol. 92. Amer Mathematical Society, 1997.

[25] Crandall, D., Owens, A., Snavely, N., and Huttenlocher, D., "Discrete-Continuous Optimization for Large-Scale Structure from Motion," in *IEEE Conf. on Computer Vision and Pattern Recognition*, 2011.

[26] Davis, T., *Direct Methods for Sparse Linear Systems*. SIAM, 2006.

[27] Davis, T., "Algorithm 915, SuiteSparseQR: multifrontal multithreaded rank-revealing sparse QR factorization," *ACM Transactions on Mathematical Software*, vol. 38, no. 1, 2011.

[28] Davis, T. A., Gilbert, J. R., Larimore, S. I., and Ng, E. G., "A column approximate minimum degree ordering algorithm," *ACM Transactions on Mathematical Software (TOMS)*, vol. 30, pp. 353–376, Sept. 2004.

[29] Dellaert, F. and Kaess, M., "Square root SAM: Simultaneous localization and mapping via square root information smoothing," *Intl. J. of Robotics Research*, vol. 25, no. 12, pp. 1181–1203, 2006.

[30] Dellaert, F., Carlson, J., Ila, V., Ni, K., and Thorpe, C. E., "Subgraph-preconditioned Conjugate Gradient for Large Scale SLAM," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2010.

[31] Dolan, E. and Moré, J., "Benchmarking optimization software with performance profiles," *Mathematical Programming*, vol. 91, no. 2, pp. 201–213, 2002.

[32] Duckett, T., Marsland, S., and Shapiro, J., "Learning globally consistent maps by relaxation," in *IEEE International Conference on Robotics and Automation*, 2000.

[33] Durrant-Whyte, H. and Bailey, T., "Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms," *Robotics & Automation Magazine*, Jun 2006.

[34] Eade, E., Fong, P., and Munich, M. E., "Monocular graph slam with complexity reduction," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, October 2010.

[35] Elkin, M., Emek, Y., Spielman, D., and Teng, S., "Lower-stretch spanning trees," in *Proc. 37th ACM symposium on Theory of computing*, pp. 494–503, 2005.

[36] Eustice, R., Singh, H., and Leonard, J., "Exactly sparse delayed-state filters," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2005.

[37] Fitzgibbon, A. and Zisserman, A., "Automatic camera recovery for closed or open image sequences," in *Eur. Conf. on Computer Vision (ECCV)*, 1998.

[38] Folkesson, J. and Christensen, H., "Graphical SLAM - a self-correcting map," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2004.

[39] Frese, U., Larsson, P., and Duckett, T., "A Multilevel Relaxation Algorithm for Simultaneous Localisation and Mapping," *IEEE Trans. Robotics*, vol. 21, pp. 196–207, April 2005.

[40] Gilks, W. R., Richardson, S., and Spiegelhalter, D., *Markov Chain Monte Carlo in Practice: Interdisciplinary Statistics*, vol. 2. Chapman & Hall/CRC, 1995.

[41] Golub, G. and Van Loan, C., *Matrix Computations*, vol. 3. Johns Hopkins University Press, 1996.

[42] Grisetti, G., Kummerle, R., Stachniss, C., Frese, U., and Hertzberg, C., "Hierarchical optimization on manifolds for online 2d and 3d mapping," in *Proc. IEEE International Conference on Robotics and Automation*, 2010.

[43] Grisetti, G., Stachniss, C., Grzonka, S., and Burgard, W., "A tree parameterization for efficiently computing maximum likelihood maps using gradient descent," in *Proc. of Robotics: Science and Systems (RSS)*, 2007.

[44] Grisetti, G., Kummerle, R., Stachniss, C., and Burgard, W., "A tutorial on graph-based slam," *Intelligent Transportation Systems Magazine, IEEE*, vol. 2, no. 4, pp. 31–43, 2010.

[45] Hartley, R. I. and Zisserman, A., *Multiple View Geometry in Computer Vision*. Cambridge University Press, second ed., 2004.

[46] Hestenes, M. R. and Stiefel, E., "Methods of conjugate gradients for solving linear systems," 1952.

[47] Horn, R. A. and Johnson, C. R., *Matrix Analysis*. Cambridge University Press, 1985.

[48] Howard, A., Matarić, M., and Sukhatme, G., "Relaxation on a mesh: a formalism for generalized localization," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2001.

[49] Huang, G., Kaess, M., and Leonard, J. J., "Consistent sparsification for graph optimization,"

[50] Ila, V., Porta, J. M., and Andrade-Cetto, J., "Information-based compact pose slam," *IEEE Trans. Robotics*, vol. 26, no. 1, pp. 78–93, 2010.

[51] Jeong, Y., Nister, D., Steedly, D., Szeliski, R., and Kweon, I., "Pushing the envelope of modern methods for bundle adjustment," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2010.

[52] Jian, Y.-D., Balcan, D., Panageas, I., Tetali, P., and Dellaert, F., "Support-Theoretic Subgraph Preconditioners for Large-Scale SLAM," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2013.

[53] Jian, Y.-D., Balcan, D. C., and Dellaert, F., "Generalized Subgraph Preconditioners for Large-Scale Bundle Adjustment," in *IEEE 13th International Conference on Computer Vision*, 2011.

[54] Jian, Y.-D., Balcan, D. C., and Dellaert, F., "Generalized Subgraph Preconditioners for Large-Scale Bundle Adjustment," *Outdoor Large-Scale Real-World Scene Analysis, Dagstuhl Reports*, 2012.

[55] Jian, Y.-D. and Dellaert, F., "iSPCG: Incremental Subgraph-Preconditioned Conjugate Gradient Method for Online SLAM with Many Loop-Closures," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2014.

[56] Johannsson, H., Kaess, M., Fallon, M., and Leonard, J., "Temporally scalable visual SLAM using a reduced pose graph," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, May 2013.

[57] Julier, S. and Uhlmann, J., "A counter example to the theory of simultaneous localization and map building," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2001.

[58] Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J., and Dellaert, F., "iSAM2: Incremental Smoothing and Mapping Using the Bayes Tree," *Intl. J. of Robotics Research*, vol. 31, pp. 217–236, Feb 2012.

[59] Kaess, M., Ranganathan, A., and Dellaert, F., "iSAM: Incremental Smoothing and Mapping," *IEEE Trans. Robotics*, vol. 24, pp. 1365–1378, Dec 2008.

[60] Klein, G. and Murray, D., "Parallel Tracking and Mapping for Small AR Workspaces," in *IEEE and ACM Intl. Sym. on Mixed and Augmented Reality (ISMAR)*, (Nara, Japan), 2007.

[61] Koller, D. and Friedman, N., *Probabilistic graphical models: principles and techniques*. The MIT press, Cambridge, MA, 2009.

[62] Konolige, K., "Sparse sparse bundle adjustment," in *British Machine Vision Conf. (BMVC)*, 2010.

[63] Konolige, K., "Large-scale map-making," in *Nat. Conf. on Artificial Intelligence (AAAI)*, 2004.

[64] Konolige, K. and Agrawal, M., "Frameslam: From bundle adjustment to real-time visual mapping," *IEEE Trans. Robotics*, vol. 24, no. 5, pp. 1066–1077, 2008.

[65] Koutis, I., Miller, G., and Peng, R., "A nearly-m*logn solver for SDD linear systems," in *Symp. on Foundations of Computer Science (FOCS)*, 2011.

[66] Kretzschmar, H., Stachniss, C., and Grisetti, G., "Efficient information-theoretic graph pruning for graph-based SLAM with laser range finders," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2011.

[67] Kruskal, J., "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical society*, vol. 7, no. 1, pp. 48–50, 1956.

[68] Kschischang, F., Frey, B., and Loeliger, H., "Factor graphs and the sum-product algorithm," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 498–519, 2001.

[69] Kümmerle, R., Grisetti, G., Strasdat, H., Konolige, K., and Burgard, W., "g2o: A general framework for graph optimization," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2011.

[70] Kushal, A. and Agarwal, S., "Visibility based preconditioning for bundle adjustment," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[71] Läuchli, P., "Jordan-elimination und Ausgleichung nach kleinsten Quadraten," *Numerische Mathematik*, vol. 3, no. 1, pp. 226–240, 1961.

[72] Lehoucq, R., Sorensen, D., and Yang, C., *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. SIAM, 1998.

[73] Leonard, J., Durrant-Whyte, H., and Cox, I., "Dynamic map building for an autonomous mobile robot," *Intl. J. of Robotics Research*, vol. 11, no. 4, pp. 286–289, 1992.

[74] Li, N. and Saad, Y., "MIQR: A multilevel incomplete QR preconditioner for large sparse least-squares problems," *SIAM Journal on Matrix Analysis and Applications*, vol. 28, no. 2, pp. 524–550, 2006.

[75] Liebchen, C. and Wünsch, G., "The zoo of tree spanner problems," *Discrete Applied Mathematics*, vol. 156, no. 5, pp. 569–587, 2008.

[76] Lourakis, M. and Argyros, A., "SBA: A software package for generic sparse bundle adjustment," *ACM Transactions on Mathematical Software*, vol. 36, no. 1, pp. 1–30, 2009.

[77] Lu, F. and Milios, E., "Globally consistent range scan alignment for environment mapping," *Autonomous Robots*, pp. 333–349, Apr 1997.

[78] MacKay, D., *Information theory, inference, and learning algorithms*. Cambridge Univ Press, 2003.

[79] Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B., "FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem,"

[80] Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B., "FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges," in *Intl. Joint Conf. on AI (IJCAI)*, 2003.

[81] Ni, K. and Dellaert, F., "Multi-level submap based SLAM using nested dissection," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2010.

[82] Ni, K. and Dellaert, F., "HyperSfM," in *3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT)*, pp. 144–151, IEEE, 2012.

[83] Nister, D. and Stewenius, H., "Scalable recognition with a vocabulary tree," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2006.

[84] Nocedal, J. and Wright, S., *Numerical Optimization*. Springer Verlag, 1999.

[85] Olson, E., Leonard, J., and Teller, S., "Fast iterative alignment of pose graphs with poor initial estimates," in *Proc. of IEEE International Conference on Robotics and Automation*, 2006.

[86] Papadimitriou, C., "The NP-completeness of the bandwidth minimization problem," *Computing*, vol. 16, no. 3, pp. 263–270, 1976.

[87] Saad, Y., *Iterative Methods for Sparse Linear Systems*. SIAM, 2003.

[88] Schlick, T., "Modified cholesky factorizations for sparse preconditioners," *SIAM Journal on Scientific Computing*, vol. 14, no. 2, pp. 424–445, 1993.

[89] Schmid, C. and Zisserman, A., "Automatic line matching across views," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 1997.

[90] Sibley, G., Mei, C., Reid, I., and Newman, P., "Adaptive relative bundle adjustment," in *Robotics: Science and Systems (RSS)*, 2009.

[91] Simon, I., Snavely, N., and Seitz, S. M., "Scene summarization for online image collections.," in *Intl. Conf. on Computer Vision (ICCV)*, 2007.

[92] Smith, R. and Cheeseman, P., "On the representation and estimation of spatial uncertainty," *Intl. J. of Robotics Research*, vol. 5, no. 4, pp. 56–68, 1987.

[93] Smith, R., Self, M., and Cheeseman, P., "A stochastic map for uncertain spatial relationships," in *Proc. of the Intl. Symp. of Robotics Research (ISRR)*, pp. 467–474, 1988.

[94] Snavely, N., Seitz, S. M., and Szeliski, R. S., "Skeletal graphs for efficient structure from motion," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[95] Snavely, N., Seitz, S., and Szeliski, R., "Photo tourism: exploring photo collections in 3D," in *ACM SIGGRAPH*, pp. 835–846, ACM, 2006.

[96] Spielman, D. and Teng, S., "Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems," in *ACM Symp. on Theory of Computing (STOC)*, pp. 81–90, 2004.

[97] Steedly, D., Essa, I., and Dellaert, F., "Spectral partitioning for structure from motion," in *Intl. Conf. on Computer Vision (ICCV)*, 2003.

[98] Strasdat, H., Montiel, J., and Davison, A. J., "Real-time monocular SLAM: Why filter?," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2010.

[99] Thrun, S., Burgard, W., Fox, D., and others, *Probabilistic robotics*. MIT Press Cambridge, MA, 2005.

[100] Thrun, S., Liu, Y., Koller, D., Ng, A., Ghahramani, Z., and Durrant-Whyte, H., "Simultaneous localization and mapping with sparse extended information filters," *The International Journal of Robotics Research*, vol. 23, no. 7-8, p. 693, 2004.

[101] Thrun, S., "Robotic mapping: A survey," *Exploring artificial intelligence in the new millennium*, vol. 1, pp. 1–35, 2003.

[102] Toledo, S. and Avron, H., *Combinatorial Scientific Computing*. Chapman & Hall/CRC Computational Science, CRC Press, 2011.

[103] Tomasi, C. and Kanade, T., "Shape and Motion from Image Streams under Orthography - a Factorization Method," *International Journal of Computer Vision*, vol. 9, no. 2, pp. 137–154, 1992.

[104] Trefethen, L. and Bau, D., *Numerical linear algebra*. SIAM, 1997.

[105] Triggs, B., McLauchlan, P., Hartley, R., and Fitzgibbon, A., "Bundle adjustment - a modern synthesis," *Vision algorithms: theory and practice*, pp. 298–372, 2000.

[106] Vaidya, P., "Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners," *manuscript*, 1991.

[107] Vial, J., Durrant-Whyte, H., and Bailey, T., "Conservative sparsification for efficient and consistent approximate estimation," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2011.

[108] Zhang, F., *Matrix Theory: Basic Results and Techniques*. Springer, 2011.